Spotting Keywords in Offline Handwritten Documents Using Hausdorff Edit Distance

Mohammad Reza Ameri

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Computer Science) at

Concordia University

Montréal, Québec, Canada

June 2018

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohammad Reza Ameri**

Entitled: **Spotting Keywords in Offline Handwritten Documents Using Hausdorff Edit Distance**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Abdel Razik Sebak

_____ External Examiner
Dr. Mahmoud El-Sakka

_____ External to Program
Dr. Nawwaf Kharma

_____ Examiner
Dr. Adam Krzyzak

_____ Examiner
Dr. Ching Y. Suen

_____ Supervisor
Dr. Tien D. Bui

_____ Co-supervisor
Dr. Andreas Ficher

Approved by _____
Dr. Volker Haarslev, Graduate Program Director
Department of Computer Science and Software Engineering

July 26, 2018

Date of Defense

_____
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Spotting Keywords in Offline Handwritten Documents Using Hausdorff Edit Distance

Mohammad Reza Ameri, Ph.D.

Concordia University, 2018

Keyword spotting has become a crucial topic in handwritten document recognition, by enabling content-based retrieval of scanned documents using search terms. With a query keyword, one can search and index the digitized handwriting which in turn facilitates understanding of manuscripts. Common automated techniques address the keyword spotting problem through statistical representations.

Structural representations such as graphs apprehend the complex structure of handwriting. However, they are rarely used, particularly for keyword spotting techniques, due to high computational costs. The *graph edit distance*, a powerful and versatile method for matching any type of labeled graph, has exponential time complexity to calculate the similarities of graphs. Hence, the use of graph edit distance is constrained to small size graphs.

The recently developed *Hausdorff edit distance* algorithm approximates the graph edit distance with quadratic time complexity by efficiently matching local substructures. This dissertation speculates using Hausdorff edit distance could be a promising alternative to other template-based keyword spotting approaches in term of computational time and accuracy. Accordingly, the core contribution of this thesis is investigation and development of a graph-based keyword spotting technique based on the Hausdorff

edit distance algorithm. The high representational power of graphs combined with the efficiency of the Hausdorff edit distance for graph matching achieves remarkable speedup as well as accuracy. In a comprehensive experimental evaluation, we demonstrate the solid performance of the proposed graph-based method when compared with state of the art, both, concerning precision and speed.

The second contribution of this thesis is a keyword spotting technique which incorporates dynamic time warping and Hausdorff edit distance approaches. The structural representation of graph-based approach combined with statistical geometric features representation compliments each other in order to provide a more accurate system. The proposed system has been extensively evaluated with four types of handwriting graphs and geometric features vectors on benchmark datasets. The experiments demonstrate a performance boost in which outperforms individual systems.

Acknowledgements

Pursuing the Ph.D. program was not solely about becoming a scientist yet to become a better person to change the world to a better place. This thesis would not have approached its successful accomplishment without guidance and support of colleagues, friends, and family.

First and foremost, to whom gave me the opportunity to pursue my Ph.D. I would like to express my sincere gratitude and appreciation toward my supervisors Dr. Bui and Dr. Fischer for being such a great mentor with their expertise, and understanding. They have always been available with persistence, attitude, motivation, and enthusiasm. I appreciate granting your help and guidance during the past years. I have learned precious lessons throughout my curriculum whether in courses I have taken or in the discussions we had. I indeed included you as *we* while writing this thesis.

Thanks to my colleagues in Switzerland for welcoming me to the fascinating world of graph-based approaches. A resourceful group, who made a great scientific leap by a graph matching framework supported by top-notch publications. I am especially thankful again to Dr. Fischer for describing to me why graphs would have a bright future in pattern recognition and subsequently heading my research toward the graph-based methods. I acknowledge Dr. Riesen and Dr. Stauffer for coauthoring and guidance through this project. Working with you made it more than a research project.

The members of the committee have discussed the progress and gave valuable input on prior stages of this research. I would like to acknowledge the Committee members Dr. Kharma, Dr. Krzyzak, Dr. El-Sakka, and Dr. Suen, for their interest in this work, valuable feedback, careful assessment, and the helpful suggestion for the improvement of this dissertation. Thanks for attending this thesis. Your insights have helped to develop the final version of this thesis.

I would like to thank CENPARMI staffs for their constant support. IMDS software has been the industrial associate and supporter of this research. I would like to thank my colleagues at IMDS who have been supporting and maintained a pleasant working environment. I would like to especially thank Dr. Ponson, the Vice President of R&D at IMDS software, and Dr. Haji for support and being there for me. Your ideas and advice on my research and my career have been exceptionally invaluable. I could not have wished for better to spend my time to enrich my experiences. The experience in the company had contributed to shaping my vision of industrial research and development of suitable software. It indeed will have a lasting impact on my profession.

I would like to warmly thank my dear friends and family for their help in moments of difficulty. For the friend who stood behind me, words cannot express how grateful I am. Thank you, mom and dad for all sacrifices, you made. Thanks to my brother for constant encouragements. Lastly, I am grateful to my beloved wife Mina for unconditional support and love and the excellent support. Thank you.

Dedicated to my beloved wife Mina

Contents

List of Figures

List of Tables

List of Abbreviations

ACC	**Acc**uracy
AK	**A**lvermann **K**onzilsprotokolle
BOT	**Bot**any
BP2	**Bip**artite **2**
BSH	**B**lurred **S**hape **M**odel
CNN	**C**onvolutional **N**eural **N**etworks
CRBM	**C**onvolutional **R**estricted **B**oltzmann **M**achine
DTW	**D**ynamic **T**ime **W**arping
DoG	**D**ifference **o**f **G**aussian
EM	**E**xpectation **M**aximization
EPC	**E**dit **P**ath **C**ost
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
GED	**G**raph **E**dit **D**istance
GMM	**G**aussian **M**ixtures **M**odel
GW	**G**eorge **W**ashington
HEC	**H**ausdorff **E**dit **C**ost
HED	**H**ausdorff **E**dit **D**istance
HMM	**H**idden **M**arkov **M**odel
HOSVD	**H**igher **O**rder **S**ingular **V**alue **D**ecomposition
HoG	**H**istograms of **O**riented **G**radients

ICA	**I**ndependent **C**omponent **A**analysis
KWS	**K**ey**w**ord **S**potting
LGH	**L**ocal **G**radient **H**istogram
LSAP	**L**inear **S**um **A**ssignment **P**roblem
LSTM	**L**ong **S**hort **T**erm **M**emory
MCS	**M**ultiple **C**lassifier **S**ystem
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
PAR	**Par**zival
PCA	**P**rincipal **C**omponent **A**analysis
PHOCNet	**P**yramidal **H**istogram **O**f **C**haracters **Net**work
PHOC	**P**yramidal **H**istogram **O**f **C**haracters
QAP	**Q**uadratic **A**ssignment **P**roblem
RBF	**R**adial **B**asis **F**unction
RNN	**R**ecurrent **N**eural **N**etworks
ROC	**R**eceiver **O**perating **C**haracteristic
SD19	**S**pecial **D**atabase **19**
SVD	**S**ingular **V**alue **D**ecomposition
SVM	**S**upport **V**ector **M**achine
TN	**T**rue **N**egative
TP	**T**rue **P**ositive
UML	**U**nified **M**odeling **L**anguage

Chapter 1

Introduction

This chapter presents a review of pattern recognition methods outlining statistical and structural categories. Then, document analysis is discussed from a pattern recognition point of view, considering both holistic and character-based recognition. Finally, motivations, objectives, contributions, and outline of the dissertation are presented.

1.1 Pattern Recognition

Pattern recognition is one of the most prominent abilities of the human being. The understanding of sophisticated patterns has helped humans to survive and as a result our cognitive and neural system evolved to make us superior in identifying patterns (Duda, Hart, & Stork, 2000). The pattern recognition field aims at giving machines the ability to determine the categories of patterns. Thereby *pattern*, an observation in the real world, is recognized by the computer to aid humans in automating an ever growing number of tasks.

The field of pattern recognition comprises a large number of topics ranging from the recognition of signature, handwriting, to the identification of objects and patterns such as human faces. These examples reveal the importance of pattern recognition for humans in

daily life. Some tasks, yet, are tedious for humans, such as sorting mails in post offices, reading the bank cheques, and recognizing cars license plates. Moreover, some tasks that are designed for machines such as computer-aided diagnostic in medical images, speech recognition for personal assistance devices, biometrics authentication such as fingerprints and faces. Thus, an algorithmic approach to the above problems requires us to use computers.

To delegate the work to machines, we must empower them with the ability to mimic human perception and intelligence. In computer science, yet, the tasks are carried out by numerical calculation. Thus the aim of pattern recognition as a field of computer science is to build mathematical models and methods to define and delegate the tasks in computer-understandable forms.

Many applications in computer science are *theory driven*. For a specific task, the precise step by step approach is given to solve the problem. *Learning-based approaches*, on the other hand, enable the machines to adapt to specific tasks instead of instructing the exact requirements. The pattern recognition problems are often too complex to have analytical solutions that provide a precise instruction for identifying a pattern. The analytical solution needs humans to analyze each model individually and make a set of instructions We would like the computers preferably, with learning-based approaches, to learn the concepts of class or category. Therefore, to recognize patterns, we build an intelligent system first to learn and then to identify the patterns.

Showing a few pictures of an unfamiliar animal, a child is able to recognize that animal in the wild. Pattern recognition aims to empower the machine with similar abilities. With this approach, machines can imitate the human's behavior of identifying objects by samples. In most pattern recognition tasks, a set of samples are first provided as examples for a specific class. The algorithm adjusts to the objects and their particular features from few examples. While adapting to a specific pattern, the algorithm nevertheless

must generalize the concept to unseen models of the same category. Then the computer employs the adapted models to classify unknown and unlabeled objects.

Pattern recognition has become one of the most demanding fields in computer science. With the effort of scientists, several problems have practical solutions to some extent. Examples include are mail sorting (Lu & Tan, 2002), spam filtering (L. Zhang, Zhu, & Yao, 2004), handwritten text recognition (Zimmermann, Chappelier, & Bunke, 2006), writer identification (Schlapbach & Bunke, 2008), and identification of persons by fingerprint (Yager & Amin, 2004), to name just a few. Although the pattern recognition approaches have succeed in these tasks, there is still room for improvement. Without doubt, with ever growing digital contents, new applications will emerge.

1.2 Statistical and Structural Pattern Recognition

The description of patterns for computers is considered to be a crucial task in pattern recognition. The representation is the underlying data structure that is used by the algorithm. Data structures provide abstractions for computers to store values, assert possible relations, and mechanism for accessing and modifying data. The choice of data structure, therefore, attributes to the general functionality of the models. Pattern recognition, based on the descriptors, is categorized as *statistical* or *structural*. The possibility of combining the two methods, as a third hybrid approach, has been investigated in Olszewski (2001). Figure 1.1 illustrates representing a word image with a graph and a sequence of feature vectors as structural and statistical representation, respectively.

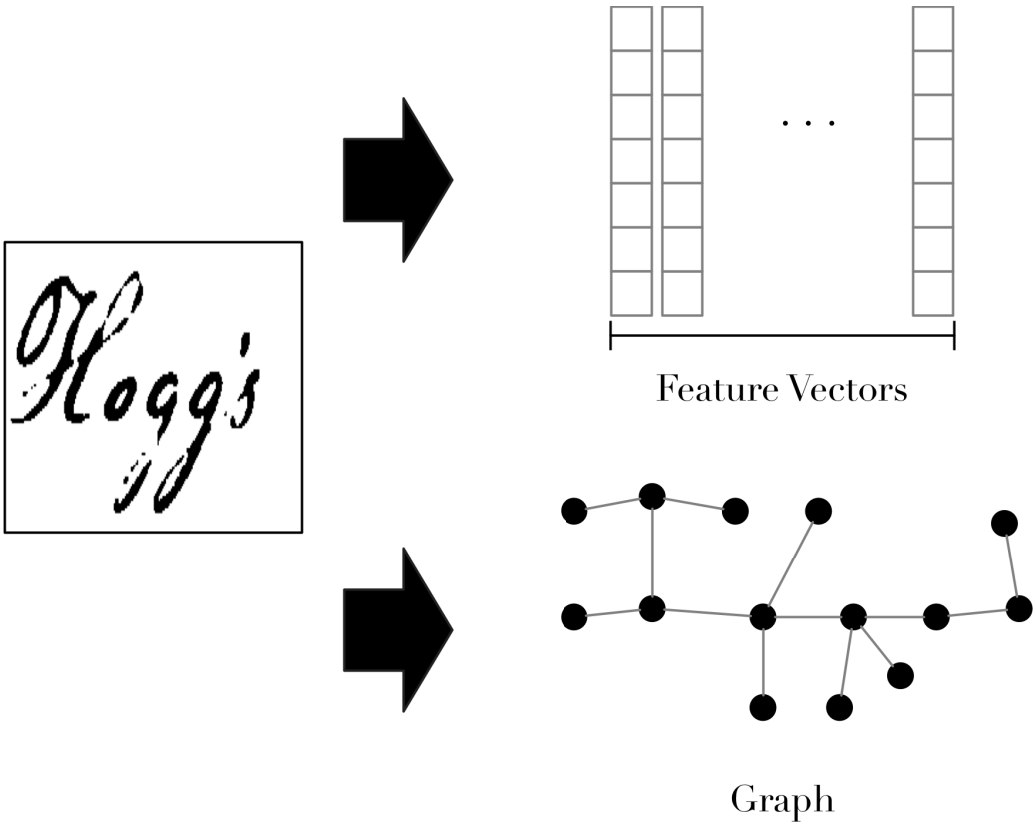


FIGURE 1.1: Representing a word image with statistical feature vectors and structural graph presentation.

1.2.1 Statistical Pattern Recognition

A feature is an n-dimensional tuple of real-valued numbers, i.e. $X = (x_1, \ldots, x_n) \in \mathbb{R}^n$. The representation of feature X is inherently a *point* in n-dimensional space. The representation often termed as *feature vector*, nonetheless, a vector defines the difference between two points. Subtracting the first point from the second one yields a vector. The concepts of points and vectors are interchangeable in pattern recognition (Pekalska & Duin, 2005). Statistical pattern recognition refers to representing the objects with fixed size vectors. The neural network approaches such as *long short term memory (LSTM)* (Frinken, Fischer, Baumgartner, & Bunke, 2014), (Frinken, Fischer, Manmatha, & Bunke, 2012), (Sankaran & Jawahar, 2012) and *convolutional neural networks (CNN)* (Sudholt & Fink, 2016) are also considered as statistical pattern recognition models.

The feature vector representation is a computationally efficient approach since modern computers internally perform vectorial operations by design. The efficiency in low computational complexity is supported by the fact that features typically are represented in the similar array-shaped data structures of computers. Moreover, the theoretical foundation of linear algebra provides extensive predefined vectorial operations. Thus, the largest number of algorithmic approaches for pattern recognition has been developed in the field of statistical pattern recognition (Duda et al., 2000).

By definition, using vectors implies a fixed-size representation even when the shape and size of different objects are not the same. Consequently, there are no means to represent the relations between different parts of a pattern. The structural methods provide a better representation of the patterns when the dependencies of substructures are more prominent than local distributions (Bunke, 1993).

1.2.2 Structural Pattern Recognition

Structural pattern recognition focuses on symbolic types of data such as graphs or strings. Graphs are defined by a set of nodes that are connected by edges. Graphs can represent patterns of variable size as well as almost any structure with the binary relation between substructures (Conte, Foggia, Sansone, & Vento, 2004). By overcoming the feature vectors drawbacks to describe structures, graphs have become a matter of interest in the pattern recognition community (Kandel, Bunke, & Last, 2007).

The high representational power of graphs, however, is accompanied by drawbacks in pattern recognition applications. The problem arises from flexibility and computation operations on graphs. For feature vectors, the comparison of two vectors is a linear-time operation concerning the size of the vectors. Graph comparison by employing graph matching often has an exponential time complexity. Moreover, even simple vector operations like sum or product are not defined in a standard way for graphs in general.

Often these operations address an individual problem rather than a general computational concept for graphs.

Whether we use statistical or structural pattern recognition, the fundamental rule is the object from the same class should have a similar feature vector or structural representation. Likewise, to have a useful pattern recognition paradigm, the feature vectors or graph-based representations must be far away and distinctly dissimilar for different objects.

1.3 Handwritten Document Analysis

Handwritten documents have been means of communication and documentation since the beginning of civilization. In the digital era, document analysis becomes a demanding task to manage and recognize digitized documents. Examples of such documents are envelopes, bank cheques, forms, manuscripts or a part of a book. The purpose of document analysis by computers, however, is not only to recognize the context yet to process documents based on the contents. The extract information could be used to sort the mails for the post offices, automate depositing cheques, and search in an extensive database of documents for specific words.

Machine-printed parts of documents are more accessible than handwritten parts. The characters in printed documents have a clear boundary and monotonic shapes. Likewise, characters have a fixed shape for a specific typeface in the document. Thus, the obstacles to recognizing a machine printed document are mostly related to scanning quality and existence of noise. The difference for a specific class of character is eventually limited to the typeface such as font, size, and orientation. The handwritten parts of documents, however, in addition to the mentioned difficulties, must cope with obstacles such as degraded characters, skewed text lines and the connected nature of cursive handwriting.

Modern digital computing devices, digital pen, and touch-sensitive surfaces empower users to digitize the drawing. Correspondingly, handwriting recognition looks beyond the scanned documents input, i.e. *offline handwriting recognition*. *Online handwriting recognition* has emerged as a second type of handwriting recognition. It aims to recognize the writing with the stylus on the surface of an electronic device. Online handwriting recognition processes the sensor input such as pen movement and pressure on the surface to translate strokes into text.

Character recognition has been the primary research interest for both handwritten and printed documents. Identifying individual characters aims at transcribing the entire documents. The digitized transcription demands the segmentation of words to the character level in advance. Such segmentation has to accommodate a higher amount of noise compared to printed characters since lexicons are often cursive in the Latin handwritten scripts. Cursive handwriting leads to ambiguity of the character boundaries. Thus, transcription of handwritten documents through character recognition does not yield a comparable result as it does for the machine printed documents.

By searching for a particular keyword, we can make the digitized manuscript more accessible. *Keyword spotting* refers to querying for a specific keyword in the documents. The system then must respond whether the keyword exists; and report the positions of the keyword subsequently. The user finally receives a list of documents with the location of the corresponding keyword. Keyword spotting was first employed in the domain of speech recognition (Myers, Rabiner, & Rosenberg, 1980) to detect a specific word in speech. Later, keyword spotting was proposed for handwritten documents by Manmatha, Han, and Riseman (1996) and Manmatha and Croft (1997).

1.3.1 Character Recognition

Initial efforts to recognize text in scanned documents focused on identifying characters as the writings building blocks. *Optical character recognition* has been an active field of research for more than three decades. Hundreds of approaches have been proposed for the recognition of handwritten characters for different scripts (Cavalin, Sabourin, Suen, & Britto Jr., 2009).

For machine-printed Latin scripts, character recognition methods achieve very high recognition rates, at least when the level of noise is low (Fujisawa, 2008). When clear imaging is available, typical recognition rates for machine-printed characters exceed 99%. However, optical character recognition is prone to errors when dealing with handwritten characters. Commercial applications with near-perfect recognition accuracy are only available for restricted tasks such as bank check reading (Gorski, Anisimov, Augustin, Baret, & Maximor, 2001). In general, the problem is still considered as mainly unsolved (Bunke & Varga, 2007).

The difficulty of recognizing handwritten characters lies in the fact that each person writes in a distinct handwriting style. In the discipline of forensic science, handwriting identification and verification are based on the principle that the handwriting of two people are not alike. In fact, forensics believe that individual's handwriting is unique to themselves; therefore, they can distinguish authentic handwriting from forged one. Consequently, the recognition system must be flexible enough to adjust to the different writing styles. Even in documents produced by a single writer, a considerable amount of ambiguity must be addressed. Fig. 1.2 shows some examples of letters from the *NIST SD19* database Grother (1995) which may be mistaken with the letter *"a"* without context. In a study by Haji (2012), the authors showed that there are at least 29 pairs of letters that may have almost identical shapes in cursive Latin handwriting. We can conclude the

(a) *'a'* or *'c'*

(b) *'a'* or *'u'*

(c) *'a'* or *'Q'*

(d) *'a'* or *'w'*

FIGURE 1.2: Ambiguity of cursive characters for the character *'a'*.

number of shapes that a handwritten character can take is substantial and challenging for pattern recognition.

Handwritten character recognition classifies individual characters in the document to the corresponding alphanumeric categories. The segmented images of characters undergo a feature extraction stage. Feature extraction is a crucial primary step that determines how well the different characters are distinguishable in the respective feature space. For an early survey, we refer to Trier, Jain, and Taxt (1996). Examples of state-of-the-art feature sets include wavelet-based representations of low-quality printed characters as well as handwritten characters Chen, Bui, and Krzyzak (2003); X. Wang, Ding, and Liu (2005);

Wunsch and Laine (1995).

It is common to preprocess feature vectors for normalization. Furthermore, there are methods for feature space transformation that apply to any feature set and may be able to improve the class separability. In general, it cannot be predicted whether the feature sets perform better with preprocessing unless it is examined in a specified recognition task. Exemplary techniques for cursive handwriting include the use of *principal component analysis (PCA)* and *independent component analysis (ICA)* Vinciarelli and Bengio (2002) as well as non-linear kernel PCA Fischer and Bunke (2009).

Recent advances in image representation include the development of sparse representations, which have proven successful for various applications in computer vision and pattern recognition Wright et al. (2010). The underlying idea is to describe an image as a linear combination of representative samples such that only a few coefficients of the linear combination are non-zero. PCA creates an orthogonal space with the aim to minimize the basis vectors. Sparse coding instead uses an extensive dictionary of representative samples to create an overcomplete basis. Following this procedure, semantic information like class membership can be directly propagated from non-zero coefficients.

1.3.2 Keyword Spotting Systems

In recent years we have seen increasing efforts worldwide by libraries and archives to digitize handwritten historical documents. Digital files of handwriting are more accessible, to search and index, by the textual content of the documents. To integrate scanned manuscript images into digital libraries based on their content, automatic handwriting recognition is needed. However, modeling and identification of handwriting are far more challenging than optical character recognition for printed text. The difficulty arises mainly due to the variable character and word shapes. Facing ancient scripts and languages, automatic transcription is often not feasible because of a lack of training data.

In most recognizer systems it is necessary to segment the text lines into characters, then a recognition system gives the proper transcription. This method works for machine printed text where character boundaries are clear, and the characters themselves are not distorted. However, for handwritten document recognition, a significant amount of ambiguity is faced in segmentation; see Figure 1.2. For such situations, *keyword spotting (KWS)* offers an alternative to index scanned manuscripts without performing a complete transcription (Manmatha et al., 1996; T. Rath & Manmatha, 2003). So-called holistic approaches consider the whole word as a unit to be recognized. Considering a whole word can effectively reduce the ambiguity which we face in individual character recognition.

Depending on the type of input handwriting, KWS operates either on online or offline handwriting. As mentioned above the online approach makes use of additional temporal information. Therefore, the additionally recorded data support the recognition of text. Hence, offline methods are typically regarded as more difficult because they operate on scanned images of documents. This thesis contributes to offline approaches of KWS where the input is given by scanned document pages.

The keyword spotting algorithms respond to the question of whether a keyword exists in the documents or not. Once the instance(s) of keyword are spotted, they are reported with their locations in the text. The query could be a string of characters or an image segment representing the keyword. The choice of the query type is not always arbitrary, because having a string-based word representation requires knowledge about the language and its alphabets. When the information about the language is not sufficient, in an ancient historical manuscript, there is no other choice than using a template image of the keyword as a query. This approach is called *template-based* keyword spotting.

Keyword Spotting Applications

Keyword spotting has several applications; here we discuss some of them briefly. In T. M. Rath, Manmatha, and Lavrenko (2004) historical documents are in the center of interest proposing a search engine for a historical documents. Keyword spotting is the primary tool used in this proposed system. Searching among historical documents is essential for libraries that digitized their books. Indexing the digitized manuscript is also another subject of interest which needs keyword spotting.

Another application of handwriting recognition is processing handwritten forms. The forms can be classified with specific keywords and transferred to the desired departments. A more sophisticated application is processing of incoming letters or parcels. In a company or organization where a lot of mail is received daily, keyword spotting can help to sort and dispatch correspondences to the related department. For example, a mail which contains the keyword *"cancellation"* or *"cancel"* is probably for terminating a contract or subscription. Thus an automated dispatch system transmits them to the corresponding department.

Keyword Spotting Methods

Early approaches of KWS employed the segmented image of keywords as query. The query image is then aligned with the word images pixel-by-pixel (Manmatha et al., 1996). The Scott and Longuet-Higgins algorithm (Scott & Longuet-Higgins, 1991) compares the template query image using affine transformations with the potential word images in documents. In the same way Leydier, Lebourgeois, and Emptoz (2007) applies the transform on zones of interest rather than pixels.

The single pixels are prone to noise particularly in handwritten documents. Later on, different feature descriptors have been investigated. The sequence of features represent

characteristics of images such as projection profiles (Manmatha & Rath, 2003; T. Rath & Manmatha, 2007; B. Zhang, Srihari, & Huang, 2003), histograms of oriented gradients (HOG) (Rodriguez & Perronnin, 2008; Rusiñol, Aldavert, Toledo, & Lladós, 2015; Terasawa & Tanaka, 2009), contours (Adamek, O'Connor, & Smeaton, 2006; Can & Duygulu, 2011), and features extracted from unlabeled data by deep neural networks (Wicht, Fischer, & Hennebert, 2016), to name just a few. The image descriptors of classic image processing like Gabor (Cao & Govindaraju, 2007) and local binary patterns (Dey, Nicolaou, Llados, & Pal, 2016; Kovalchuk, Wolf, & Dershowitz, 2014) and scale invariant feature transform (Konidaris, Kesidis, & Gatos, 2015), are applied in KWS problems as well. Another well-known descriptor proposed by Marti and Bunke (2002) includes nine geometric features.

For coping with the variable width of the handwriting, a widely adopted approach is to use a sliding window for extracting a sequence of feature vectors from word images and match two sequences by means of dynamic time warping (DTW) (T. Rath & Manmatha, 2007). To avoid an explicit segmentation of the scanned document page into word images, segmentation-free methods have been proposed as well (Rusiñol et al., 2015).

Two general approaches to keyword spotting can be distinguished, viz. *template-based* and *learning-based* methods. While template-based methods match one or several instances of a keyword image directly with the scanned manuscript, learning-based methods aim to learn word or sub-word models from labeled training samples. Examples include learning with hidden Markov models (HMM) (Fischer, Keller, Frinken, & Bunke, 2012; Perronnin & Rodríguez-Serrano, 2009; Rothacker & Fink, 2015), support vector machines (SVM) (Almazan, Gordo, Fornes, & Valveny, 2014), recurrent neural networks (RNN) (Frinken et al., 2012), and convolutional neural networks (CNN) (Sudholt & Fink, 2016). In general, learning-based methods are able to achieve a significantly better performance than template-based methods. However, they are less flexible because they

require a considerable amount of labeled training data. The template-based paradigms, on the other hand, requires no knowledge of underlying languages of documents.

Two general limitations of feature vector descriptors relate to their representational power. Firstly, they have to capture the structure of handwriting with a fixed number of real-valued features regardless of the complexity of the given instance. Secondly, they cannot represent binary relations between parts of the handwriting in a straight-forward way. Both limitations can be solved by means of graph-based representations which model parts of an object with nodes and relations between the parts with edges (Conte et al., 2004). In recent work, several graph-based methods have been proposed in the context of template-based keyword spotting, using keypoints as nodes (Howe, 2013; Stauffer, Fischer, & Riesen, 2016a; P. Wang et al., 2014a, 2014b) or basic strokes as nodes (Bui, Visani, & Mullot, 2015; Riba, Llados, & Fornes, 2015), and connecting them with edges if there is a connection in the image.

The main drawback of graphs, however, is that their high representational power comes at the cost of high computation complexity. Most of the aforementioned methods for graph-based keyword spotting use the well-known bipartite approximation (BP) (Riesen & Bunke, 2009a) of the graph edit distance (GED) (Bunke & Allermann, 1983). Although BP reduces the $\mathcal{NP}$-complete problem of GED to a polynomial-time assignment problem, it still has a cubic time complexity with respect to the graph size, which imposes significant computational constraints for keyword spotting.

1.4 Motivations

Pattern recognition is an inherent ability of human beings. This ability has helped us to survive in daily life. However, some pattern recognition tasks are quite tedious for us and, accordingly, there is an increasing interest to delegate such tasks to machines.

In order to empower machines to solve the pattern recognition problems, we must implement the problem in mathematical and algorithmic forms. The formulation of patterns typically falls under structural or statistical representation. Regarding the benefits and drawbacks of each method, the selection must be carefully made.

The area of interest in our research is the recognition of handwritten documents. This thesis has been inspired with our earlier investigation in the handwritten document recognition:

- Holistic keyword spotting approach (Haji, Ameri, Bui, Suen, & Ponson, 2014).
- Segmented-based character recognition (Ameri, Haji, Fischer, Ponson, & Bui, 2014).

Based on our investigation the character-based approach for handwritten document analysis has two pitfalls:

(1) Errors in the segmentation of cursive handwriting.

(2) Difficulty in the identification of characters outside the word context.

Therefore, we focus on holistic keyword spotting approaches.

The keyword spotting in historical manuscripts aids libraries to provided the annotated versions of their collections. Users can retrieve documents which are related to a search query within the collections. The string-based query requires a prior knowledge of the language of the document which is not always applicable in ancient manuscripts. The template-based query is independent of underlying language. Therefore, the template-based approach better suits to the situation.

In the statistical approaches, we employ fixed-size feature vector to represent the characters and sequence of fixed-size vectors to represent words. We have speculated whether

the statistical representation cannot preserve the essential information in order to recognize documents with more accuracy. The structural representation has the flexibility to represent objects of different shapes and sizes. In particular, they are pretty useful when the dependency of substructures can be used toward discriminating patterns. However, these representations suffer from a lack of efficient computational algorithms. Statistical approaches have a large number of learning-based algorithms at their disposal for classification tasks; nevertheless, they risk losing discriminative information of the original objects related to dependencies among substructures.

1.5 Objectives

In this thesis, we investigate the potential of a recently introduced more efficient approximation of GED, namely the Hausdorff edit distance (HED) (Fischer, Suen, Frinken, Riesen, & Bunke, 2015). It has a quadratic time complexity with respect to the graph size similar to DTW, which has a quadratic time complexity with respect to the sequence length. Unlike DTW, HED is not constrained to sequence matching. Instead it is able to match arbitrary handwriting graphs without constraints as regards the graph structure and the label alphabets for nodes and edges.

Our objective is to develop the keyword spotting system which is:

Holistic Keywords are spotted as a whole.

Offline Spotting keywords on scanned documents.

Template-based The query can be arbitrary image segments in the document.

Structural Using the graph representation of handwriting.

Fast Investigating the quadratic time algorithm rather than the cubic time approximation of GED.

1.6 Contributions

The contributions of this thesis have been presented and published in:

GBR Conference Ameri, Stauffer, Riesen, Bui, and Fischer (2017).

Pattern Recognition Letter Journal Ameri, Stauffer, Riesen, Bui, and Fischer (2018).

The main contribution of this thesis is a keyword spotting paradigm. The proposed system considers graphs from the beginning to the end for representation and distance computation. The second contribution proposes the hybrid keyword spotting system which combines structural and statistical representation in a hybrid HED-DTW-based keyword spotting system.

In this thesis, we put forward the idea of using the efficient quadratic-time HED algorithm for the purpose of calculating the dissimilarity of handwriting graphs.

The proposed systems have the following properties:

Learning Free It has few parameters that are optimized.

Transferable Parameters Although we optimize the system on a particular batch of documents, transferring these default parameters can lead to comparable performance.

The combined system additionally benefits from the statistical and structural approach in one system.

The contributed approaches have been evaluated on several benchmark datasets of historical manuscripts. The results demonstrate:

- Superior performance improvement with respect to the **accuracy** of results.
- A significant **speedup** compared to the BP-based KWS.

when compared with other template-based keyword spotting techniques regarding speed and accuracy.

1.7 Outline

The remainder of this thesis organized as follows. In chapter 2 different approaches to keyword spotting in handwritten documents are reviewed and discussed. Afterward, graph-based representation and graph matching algorithms are presented in chapter 3. Chapter 4 is dedicated to the *graph-based keyword spotting* using the *PB algorithm*. The structural representation of handwriting with graphs and the architecture of the system are described.

The contributions, proposed approaches to keyword spotting, are then presented and empirically evaluated in chapter 5. Besides the HED-based keyword spotting system, the idea of the multiple classifier systems is put forward. It combines both statistical and structural techniques for keyword spotting, profiting from their complementary perspectives on the handwriting. Through comprehensive experiments, we demonstrate a promising performance of the proposed method on various historical handwriting benchmark datasets, with respect to both accuracy and computational efficiency. Finally, chapter 6 concludes the thesis and highlights a potential path for future lines of research.

Chapter 2

Keyword Spotting in Handwritten Documents

Keyword spotting systems can be broadly categorized into *template-based* and *learning-based* approaches. The former paradigm involves either structural or statistical representation to match template images of the keyword with document images. The template-based methods, when a statistical representation is used, are often based on the *dynamic time warping (DTW)* algorithm. In this case, DTW is employed for alignment as well as for computation of the pattern's dissimilarities.

The latter paradigm is typically based on statistical machine learning algorithms; hence, they operate on vectorial representations. The learning-based approaches utilize a variety of statistical learning algorithms such as *hidden Markov model (HMM)* (Fischer et al., 2012; Perronnin & Rodríguez-Serrano, 2009; Rothacker & Fink, 2015), support vector machines (SVM) (Almazan et al., 2014), recurrent neural networks (RNN) (Frinken et al., 2012), and convolutional neural networks (CNN) (Sudholt & Fink, 2016).

HMMs are widely used for recognition of sequential patterns, such as series of characters or strokes in a handwritten document. A typical segmentation-based workflow for keyword spotting is illustrated in Figure 2.1.

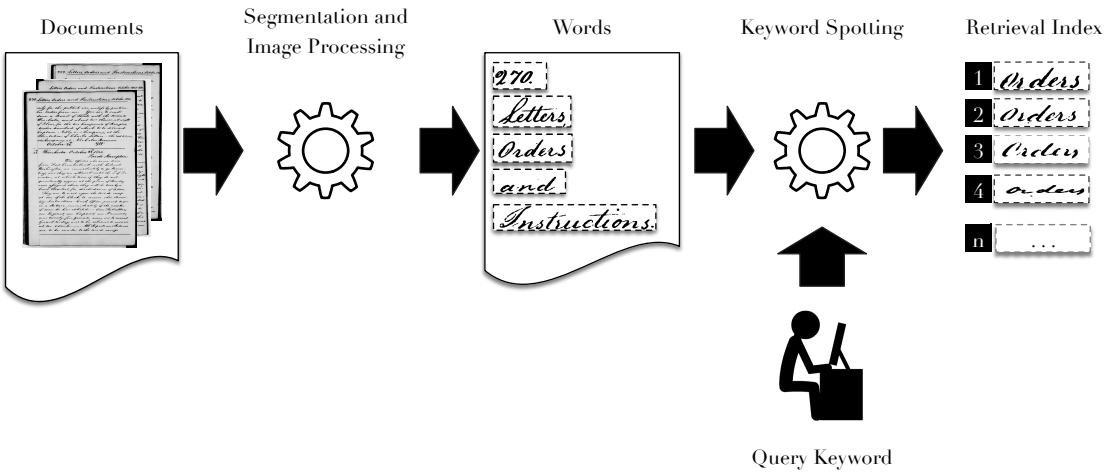


FIGURE 2.1: Keyword spotting workflow for document indexing.

In this section, we describe both template-based KWS with DTW and learning-based KWS with HMM in more detail, in order to present examples of well-established spotting methods based on statistical feature vector representations. The HMM-based system has been implemented by the author of the present thesis in the context of an industrial application.

2.1 DTW in keyword spotting

Keyword spotting was first introduced in Myers et al. (1980) in the speech recognition community. It was introduced to the field of document analysis by Agazzi (1994). Later on, Manmatha et al. (1996) employed DTW for indexing historical documents by a template-based keyword spotting approach. In this section, the method proposed by T. Rath and Manmatha (2007) is explained in more detail.

Spotting keywords in digitized documents by means of the template-based approach starts with a preprocessing step that includes segmenting and correcting skew and slant of the words. Then a feature extraction step converts the images into feature vectors sequences. The distance between these feature vectors sequences is the measure of similarity of

the compared words. The distance is computed by means of the DTW algorithm which aligns two vector sequences and returns the minimum cost of such an alignment. The DTW algorithm used in in a variety of recent KWS approaches like Adamek et al. (2006); Frinken et al. (2012); Wicht et al. (2016).

2.1.1 Feature Extraction

To calculate the dissimilarities, the DTW-based keyword spotting system performs two crucial stages: feature extraction and alignment. The feature extraction accounts for converting the images to a particular vectorial representation. The purpose of feature extraction here is to vectorize the two-dimensional image. The vector elements contain higher-level information than the pixel values and temporal information in the reading direction.

A simple yet efficient feature extraction method has been proposed by Marti and Bunke (2002). The feature vector is a collection of nine geometric features. The word image is processed from left to right direction with a sliding window. The first three features measure the global aspect of a window: the weight of the window (fraction of foreground pixels), the center of gravity, and second-order moment. Feature four to nine process the writing style. Upper bound, lower bound, and their derivatives represent the boundary information. The number of foreground-to-background transitions and the fraction of foreground pixels within the boundary complete the list.

An arbitrary image I, extracted from a handwritten document, has a height of h and width of w. The pixel value at column c and row r of the image is represented by $I(r,c)$. It is either 1 (foreground, black) or 0 (background, white) after binarization. The above approach extracts a sequence of w feature vectors. Each vector corresponds to a sliding window of size one that moves from left to right on the image.

The feature vector F is the combination of the nine features $f_1, ..., f_9$:

$$F = (f_1, ..., f_9) \tag{2.1}$$

The feature vectors further normalized to $\hat{F}$ with z-score where F_μ and F_σ are mean and variance , respectively, computed over the whole feature vector sequence of the handwriting image.

$$\hat{F} = \frac{F - F_\mu}{F_\sigma} \tag{2.2}$$

2.1.2 Distance Computation by DTW

To compare the distance of two sequences of feature vectors the straightforward approach is resampling the sequences of vectors to have the same number of samples. Therefore, the distance can be computed by the Euclidean distance of corresponding elements. This matching does not consider the variation in handwritten text and assumes the corresponding points are located at precisely the same positions. DTW aligns two time series in order to find the corresponding points on the time axis. Such points can be located at different times. This allows the nonlinear sequence alignment by compressing and expanding the time axis.

For vector sequences $X = (x_1, ..., x_M)$ and $Y = (y_1, ..., y_N)$ the DTW distance, $\text{dist}(X, Y)$, is computed with the help of a dynamic programming cost matrix $D \in \mathbb{R}^{M \times N}$. $D(i, j)$ indicate the alignment cost of two sub-sequences $X_{1:i}$ and $Y_{1:j}$. Intuitively $D(M, N)$ is the cost of the complete mapping of X and Y.

$D(i,j)$ is recursively computed by :

$$D(i,j) = \min \left\{ \begin{array}{c} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{array} \right\} + d(x_i, y_j) \tag{2.3}$$

$d(x_i, y_j)$ can be the squared Euclidean distance of the d-dimensional feature vectors x_i and y_i as indicated by Eq. 2.4.

$$d(x_i, y_j) = \sum_{p=1}^{d} (x_{i,p} - y_{j,p})^2 \tag{2.4}$$

The DTW algorithm is described in Algorithm 1.

Algorithm 1 DTW Algorithm

Require: $X = (x_1, ..., x_M)$, $Y = (y_1, ..., y_N)$ and distance function $d(.,.)$ i.e. Euclidean distance.
Ensure: DTW distance
1: $D(1,1) \Leftarrow d(x_1, y_1)$
2: **for** $i \in (2, M)$ **do**
3: $\quad D(i,1) \Leftarrow D(i-1,1) + d(x_i, y_1)$
4: **end for**
5: **for** $j \in (2, N)$ **do**
6: $\quad D(1,j) \Leftarrow D(1,j-1) + d(x_1, y_j)$
7: **end for**
8: **for** $i \in (2, M)$ **do**
9: $\quad$ **for** $j \in (2, N)$ **do**
10: $\quad\quad D(i,j) \Leftarrow \min \left\{ \begin{array}{c} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{array} \right\} + d(x_i, y_j)$
11: $\quad$ **end for**
12: **end for**
13: **return** $D[M,N]$

2.2 HMM in Handwritten Keyword Spotting

Learning-based approaches to KWS employ statistical learning algorithms. Such algorithms train a distinct number models *a priori* in the training stage. The models learn the characteristics of patterns from the training data to categorize words within corresponding classes. HMMs are one of the most widely used approaches to develop such models. Training HMM models on character images, Edwards et al. (2004) have proposed a system to transcribe Latin manuscripts. Another example is the HMM-based approach for printed and handwritten Arabic letters proposed by Chan, Ziftci, and Forsyth (2006).

The entire word descriptor by Lavrenko, Rath, and Manmatha (2004), which combines scalar and projection-based features, highlighted a new path to use HMMs for recognizing entire words. The authors employed continuous HMM and semi-continuous HMM to model the words (Rodríguez-Serrano & Perronnin, 2009). Later on, Rodríguez-Serrano and Perronnin (2012), in the context of KWS investigated semi-continuous HMM in conjunction with DTW matching. The HMM-based method proposed by Fischer et al. (2012) follows a segmentation-free approach for character-based and lexicon-free KWS in complete text lines.

The following section describes the HMM-based keyword spotting systems developed for an industrial project that is inspired by Rodríguez-Serrano and Perronnin (2009).

Keyword spotting is an essential part of the document analysis workflow. The collection of input data is processed and segmented, and then the word matching step discovers potential keywords. Finally, based on the spotted keywords information retrieval task such as document classification can be performed as illustrated in Figure 2.1.

Keyword spotting can be used to retrieve related documents written in cursive scripts. The process considers the recognition of words segmented at the word or line level rather than individual characters. The recognition engines used for this purpose are recognizer

based on sequence alignment. The HMM is a statistical model which performs the task of aligning and matching the sequences at the same time. We intended to investigate the HMM-based recognition engine with respect to the document analysis workflow that is used to classify the documents with user-defined keywords.

Handwritten document recognition can be classified into two categories, i.e. holistic and character-based, with respect to the patterns that are modeled and recognized. In the holistic methods, the classes represent words from a given lexicon. In character-based methods, the sub-word models, i.e., characters, are the recognition units. For recognizing cursive handwriting, the holistic techniques may have a certain advantage - if enough training data is available - as they model the entire shape of a word. HMM models has been proposed to employ either of these methods. In the holistic approach, an HMM is trained to recognize a specific word. The keyword models evaluate the candidate words as accepted or rejected.The disadvantage of this method is that the number of models noticeably rise when taking words into account. Thus a limited number of words are trained in practice. Consequently, the system cannot recognize out-of-vocabulary keywords. In character-based methods, individual HMMs represent characters. Thus, arbitrary keyword models can be composed of the individual character HMMs.

2.2.1 Theory of HMMs

HMMs are directed graphical Markov models Rabiner (1989). The HMMs can predict or generate a sequence of events based on sequence of hidden states. The sequence of events are related together and visible hence they also are called observations. The probability of an event only depends on the previous one. The HMMs then compute the probability of a sequence of observations.

A sequence of observations, $O = (o_1, \ldots, o_T)$, is emitted from states $Q = \{q_1, \ldots, q_N\}$. In Figure 2.2 the states are *High pressure*(HP) and *Low Pressure*(LP) and there are two

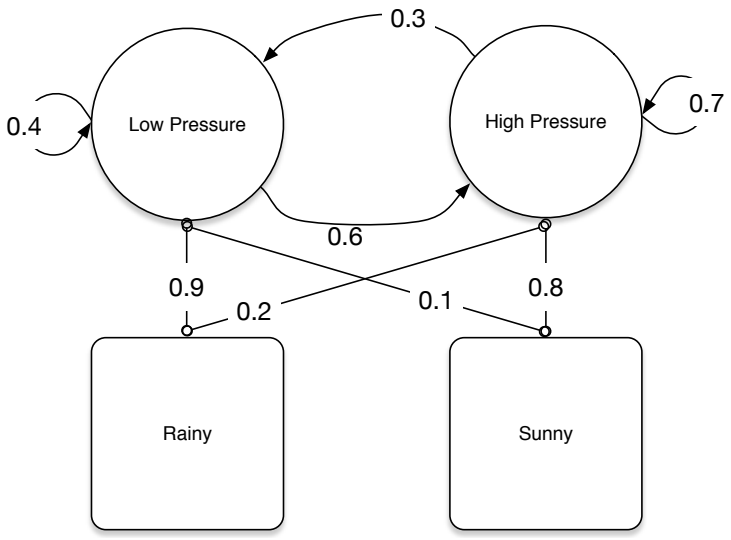


FIGURE 2.2: Hidden Markov model for predicting weather.

possible observations *rainy* and *sunny*. The HMM models provide no direct interaction with the states. For an observation the HMM, from a hidden state, transit to another connected state. The state transition probability determines which state is more likely to proceed. Once a state is reached, the probability of an observation adds up to the sequence likelihood. Being at the *LP* state, the chance of a rainy or sunny day is called emitting probability represented by $P(rainy|LP) = 0.9$ and $P(sunny|LP) = 0.1$. Next day, the atmosphere can be the same with the probability of $P(LP|LP) = 0.4$ or change with the probability of $P(HP|LP) = 0.6$.

HMM theory makes three assumptions for such inference.

- Markov assumption.
- Stationary assumption.
- Output independence assumption.

The Markov assumption states that next state only depends on the current state $P(q_{t+1}|q_t)$. The stationary assumption is about independence of the probability from the time frame that is $P(q_{t+1} = i|q_t = j) = P(q_{t+k+1} = i|q_{t+k} = j)$. Transiting from state i to j, the

probability is the same for every event. Finally the output independence assumption is about independence of the emitted probability for a sequence of observations $P(O) = \prod_{t=1}^{T} P(o_t)$.

In discrete HMM models $\lambda = (A, B, \Pi)$, A is the transition probability, Π is initial state probability, and B is the emission probability. B is represented by a $N \times M$ matrix where N is the number of states, and M is the number of observation symbols.

HMM models for continuous observation

In some applications the observations have continuous forms i.e. $o \in [0,1]$. The Continuous HMM models accept the continuous observation. The emitting probability for the states, in this case, is modeled by a continues distribution probability. Using a Gaussian distribution or more general a mixture of Gaussian distributions is a standard approach to model continuous observations in HMMs. The Gaussian function has two parameters, the mean μ and the standard deviation σ. If a mixture of Gaussian functions is used, a weight parameter c for each Gaussian is provided. Figure 2.3 shows the weather forecast with continuous observation i.e. Temperatures.

Algorithms

Three fundamental problems must be solved for HMMs: the evaluation, decoding and learning problem. The evaluation problem computes the probability of observation $O = (o_1, \ldots, o_T)$ by a given HMM model λ. The probability $P(O|\lambda)$ computes the odds of generating observation O by HMM model λ. The HMM model yet could generate O probably with more than one path through hidden states. The *Forward-Backward* algorithm calculates the probability considering all the paths within HMM state sequences that generates O. The second fundamental problem, decoding, aims at finding the most

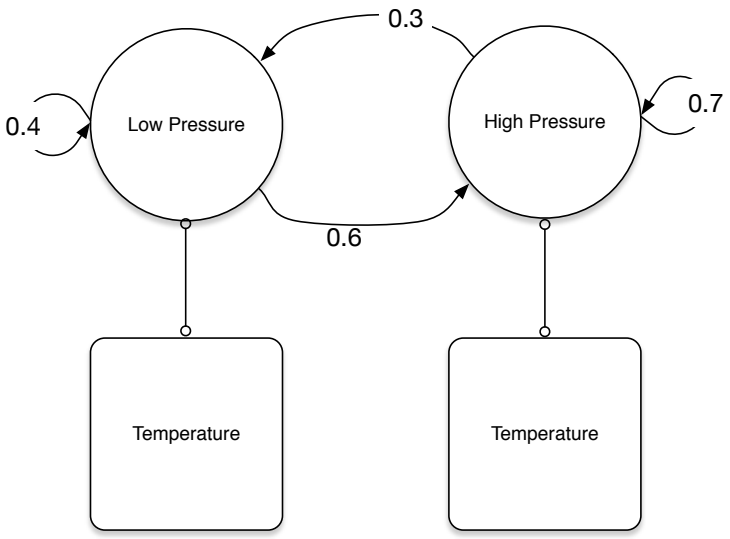


FIGURE 2.3: Hidden Markov model with continuous observation for predicting temperature.

likely sequence of states that produces the observation O. For a possible solution of this problem the *Viterbi* algorithm can be used.

The learning problem refers to training an HMM on a set of observation sequences. The *Baum-Welsh* algorithm maximizes the probability of observation O by adjusting the HMM model parameters. The algorithm re-estimates the parameters iteratively to maximize the likelihood of the observation sequence.

Chapter 3

Graph Matching Algorithms

Finding an ideal representation of objects has a direct influence on the performance of the pattern recognition systems. It is essential to capture the main characteristics of an object for any pattern recognition system to succeed. Graphs are universal data structures that can model complex objects with arbitrary interconnection of substructures. Therefore, graphs make an excellent choice to represent patterns as they can express structures of arbitrary size and complexity.

However, few pattern recognition applications adopted graph-based methods. In fact, the main disadvantage of graph-based approaches is from computational point of view. In typical feature vector representation, we can perform pairwise operations on two feature vectors in linear time. Whereas in graphs, computing some of the most basic operations such as the sum are not defined in general. In a specific framework, we can define such operations tailored to the specific application domain. The flexibility of nodes and edges of arbitrary size and order leads to additional problems of obtaining correct correspondence. Any arbitrary order can hold the optimal solution since there is no specific way known a priori for the related structures of graphs. However, the graph edit distance algorithm explores an exponential number of solutions to find the optimal one.

To find a quantitative metric for the degree of similarities in graphs, we compare the structures to find similarities in the subgraphs. Based on the associated structures, consequently, we can define a proximity measure. The task, namely *graph matching*, aims at comparing the structure of two graphs to find their resemblance. Then we define a measure of proximity that is applied to the corresponding nodes and edges to quantify the matching. The operation is a crucial process concerning the exponentially growing number of subgraphs.

Graph matching is formally categorized as *exact graph matching* and *inexact graph matching*. In the exact graph matching point of view, similar graphs require a strict mapping between nodes and edges otherwise they are considered dissimilar. In the inexact graph matching, however, the algorithms tolerate some degree of deterioration in the structures. In other words, even if two graphs are not entirely similar, they can be matched. The detected similarities and dissimilarities comprise the outcome. Error-tolerant graph matching is the preferred approach for our pattern recognition application as the patterns usually are subjected to noise and show some degree of deterioration.

3.1 Graph Definition

Graphs are fundamental concepts in math, the terminology that is used in other fields may have slight variations. The following definition, most common in the discrete math domain, provides a reliable description that we have adopted in the course of this thesis.

Definition 1 *A graph g is defined as a four-tuple* $g = (V, E, \delta, \nu)$ *where*

- V *is the set of finite nodes.*
- $E \subseteq V \times V$ *is set of finite edges.*
- $\delta \rightarrow L_V$ *is node labeling function.*

- $\nu : E \rightarrow L_E$ *is edge labeling function.*
- L_V *is the finite or infinite set of labels.*
- L_E *is the finite or infinite set of labels.*

Based on individual attributes of nodes and edges, we can categorize graphs into *labeled* or *unlabeled* graphs. In the former case, both nodes and edges have an arbitrary numerical, vectorial, or symbolic label from L_V or L_E, respectively. In the latter case, we assume empty label alphabets, i.e., $L_V = L_E = \{\}$.

Edges represent the connection between graph nodes by pair of source and target node (u,v). The nodes u and v are called *adjacent* when they are connected by an edge $e = (u,v)$. Often, the adjacency considered *directed* from the source to the target node i.e. $u \rightarrow v$. One can define the *undirected* edge that implies forward and backward adjacency, $u \rightarrow v$ and $v \rightarrow u$ respectively. Additionally, graphs can be divided into *undirected* and *directed* graphs, where pairs of nodes are either connected by undirected or directed edges.

Graphs substructures, namely subgraphs, span over a subset of graphs nodes and edges. One can make a subgraph of g by removing some of its nodes and edges. The nodes and edges labels of subgraph however are kept intact.

Definition 2 *For graphs* $g_1 = (V_1, E_1, \delta_1, \nu_1)$ *and* $g_2 = (V_2, E_2, \delta_2, \nu_2)$ g_1 *is subgraph of* g_2 *i.e.* $g_1 \subseteq g_2$ *if*

- $V_1 \subseteq V_2$.
- $E_1 \subseteq E_2$.
- $\delta_1(u) = \delta_2(u)$ *for all* $u \in V_1$.
- $\nu_1(e) = \nu_2(e)$ *for all* $e \in E_1$.

We can obtain g_1 as the *induced* subgraph of g_2 by the strict condition on the subgraph edges.

$$E_1 = E_2 \cap V_1 \times V_1$$

The subgraph therefore has less or equal number of nodes yet the entire relevant edges.

3.2 Graph Matching Definition

The process of graph matching aims at finding the similar substructure of underlying graphs. The graph matching algorithms could be categorized as *exact* and *inexact* algorithms. The exact matching algorithms only map the substructures if they are identical. The latter approach, inexact matching, tolerates some degree of dissimilarity in substructures. Therefore it always finds a bijection between graphs even when they are not alike. Based on the (dis)similarity of graphs, i.e., variation in substructures, a distance can be calculated. The distance is the proximity measure of graphs known as *graph comparison problem*. The comparison problem is defined as

Definition 3 *For graph g_1 and g_2 graph comparison problem is to find the function d on graph domain G such that*

$$d : G \times G \rightarrow \mathbb{R} \tag{3.1}$$

d is the proximity of graphs g_1 and $g2$.

Graph matching and graph comparison inherently are different concepts. Graph matching maps structure of graphs and finds an edit path from one graph to another. Graph comparison, on the other hand, calculates the distance between graphs. However, in the course of this thesis graph matching is used to find the edit map and consequently the

mapping is used to calculate the distance between graphs. Hence, graph matching and graph comparison carry the same concept and could be used interchangeably.

The exact graph matching approach asserts identical structure and labels of two graphs or at least their subgraphs. Such identity determined by graph isomorphism concept. Graph isomorphism defines a bijection function that maps g_1 to g_2. Eventually, for every node and edge in g_1, it finds a corresponding node and edge with the same label in g_2. In a less restrictive case, the *subgraph isomorphism* requires a mapping between g_1 and a subgraph of g_2.

Graph matching employing an exact approach requires exact structure and topology in corresponding graphs to recognize their similarity. Slight difference in topology or labels is interpreted as different graphs. In pattern recognition problems, objects of the same category typically do not have the same structure but having similarities. Thus graph extracted from objects in the same class are not identical. Moreover, often noise are inevitable in extraction process that can affect graph structure and labels. Consequently, the drawback of exact graph matching is with the assumption that there are no noise and deformation in the patterns. Hence the exact graph matching is rarely applied in the real-world applications.

The inexact graph matching algorithms allow different topology in graphs. They could be used in more general and broader application due to the relaxed constraint. Thus, the (dis)similarity score of objects makes broader understanding rather than the binary result, namely same or different, in the exact graph matching. Being more specific, instead of determining whether two graphs, i.e., nodes, edges, and their labels, are the same, the inexact graph matching algorithms assess the similarity with a quantitative measure. The matching cost defines a higher cost for different nodes or edges, and a lower cost for similar nodes and edges. Thus, the matching algorithm favors mapping similar structures to minimize the eventual matching cost. Furthermore, the inexact matching algorithms

do not reject the graphs with different structures, yet they penalize such a condition with a higher matching cost. Thus, nor the graph labels neither the structure of edge need to be identical. Hence, two graphs have a mapping no matter how different they are yet the matching cost describes the discrepancy.

An example in Figure 3.1 demonstrates inexact graph matching. The graph g_1 with black nodes is matched against graph g_2 with white nodes. The algorithm performs a series of operations, *deletion, substitution*, and *insertion*, to show a possible matching path between two graphs. The graphs represent the word "you" with a slight variation in writing style. The similar parts are assigned with substitution operations. Deletions and insertions of dissimilar parts in edit path determine a remarkable discrepancy.

Error-tolerant graph matching is proposed within broad range algorithms including yet not limited to the graph edit distance that uses tree search method, genetic algorithms the relaxation labeling technique, kernel methods, spectral approaches, and artificial neural networks. It is possible to use any mentioned method, in theory; however, we devote this thesis to the study of graph matching paradigms based on the graph edit distance algorithm. This paradigm is known to handle arbitrary graphs and not restricted to the particular cases. In remaining sections, we describe the *graph edit distance (GED)* algorithm with a further discussion of practical usage due to the computation time complexity. The *NP-hard* algorithm discovers the optimal solution in an exponential tree. Consequently, we justify the usage of approximation algorithms with *quadratic* and *cubic* time complexities as the alternatives.

3.3 Graph Matching in Structural Pattern Recognition

We addressed two significant classes of pattern recognition techniques: statistical and structural. The statistical pattern recognition operates on vectorial data. The process of

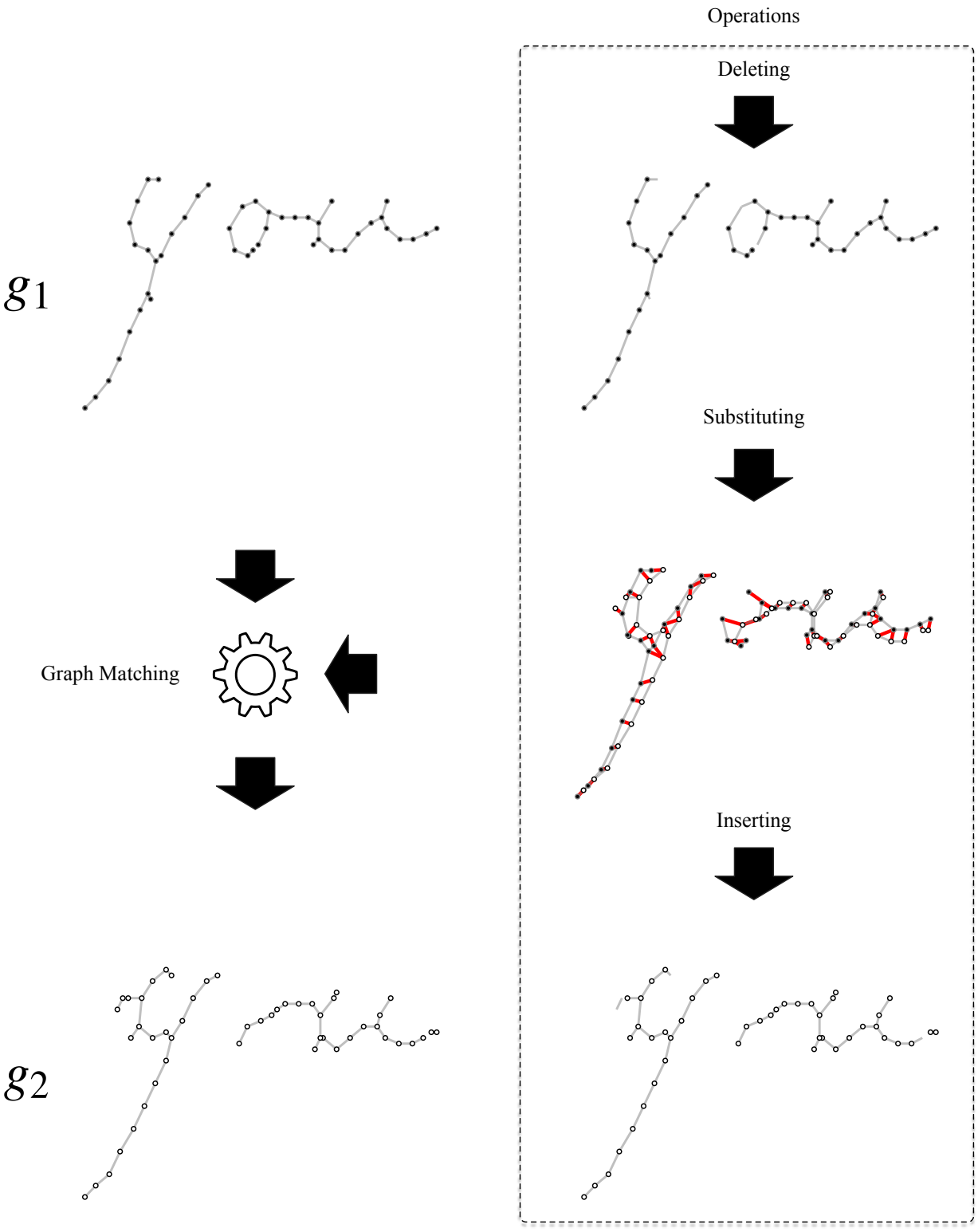


FIGURE 3.1: A possible inexact match between g_1 to g_2.

statistical pattern recognition requires extracting feature vectors. A local window often moves over a word image to extract features. The features of a word image are created in a sequence of independent vectors. More specifically, a vector holds the characteristics of a local image patch. The vectors do not provide any information about relations between substructures such as correlation with other patches. A complex structure holds many links between substructures. The feature vectors disregard the complex structure of patterns since they cannot preserve the complex structure. As a consequence, the feature vectors are not suited to represent structural information since a complex shape is presented by binary relations between subcomponents.

We need to preserve global characteristics of patterns for a complex shape. The feature vector captures independent information of local windows. The graphs can represent the patterns of variable size as well as the binary relation between substructures. Graphs could represent a various amount of structural patterns, i.e., strings, trees, or graphs. Graphs have become a matter of interest in pattern recognition community to overcome the feature vectors drawbacks.

The pattern recognition techniques often rely on similarities or dissimilarities between patterns. However, the similarity concept on arbitrary graphs has no specific corresponding unit of measurement. The *graph matching algorithms* can compare graphs and show a set of editing operations between two graphs. The editing operations have potential and flexibility to quantify the difference between graph-based patterns (Conte et al., 2004). Graph matching based pattern recognition has received considerable attention in the field of pattern recognition recently (Foggia, Percannella, & Vento, 2014).

Exact graph matching requires strict similarities between graphs hence it is not a suitable choice to compare handwriting graphs. Graph matching algorithm must be flexible enough since handwritings have considerable variations. *Inexact* graph matching paradigm, however, tolerates differences in graphs. The *graph edit distance (GED)*

algorithm, as an inexact graph matching algorithm, performs the matching by considering deletion and insertion of different subgraphs in addition to substitutions. We consider GED based matching for the keyword spotting framework as it tolerates inequality in graphs. Thus rather than the binary comparison of exact graph matching, we can quantify the matching by assigning a specific cost to the operations.

The graph matching based on the GED paradigm (Bunke & Allermann, 1983; Sanfeliu & Fu, 1983), provides a flexible quantitative measure to compare graphs. However, the primary drawback is associated with the time complexity of GED as it belongs to *quadratic assignment problems (QAP)* class of *NP-complete* algorithms. Thus GED is not applicable in most real word problems other than small sized graphs. Being *NP-complete* means polynomial solutions do not exist unless *P=NP*. The classic solution yet performs an exhaustive search in the exponential space of solutions to calculate optimal GED.

The approach to use graph matching algorithms for pattern recognition problems established by Riesen and Bunke (2009b); Riesen, Neuhaus, and Bunke (2007) is known as *bipartite graph edit distance (BP)*. The BP algorithm reduces the exponential graph matching to cubic time by considering the local structure of graphs rather than global.

The BP algorithm transforms the graph matching to the *linear sum assignment problem (LSAP)* with cubic time complexity. Hence, the BP graph matching method views the nodes of graphs as elements of sets. Since LSAP assigns equal size sets, BP adjusts the size of sets by adding null operations ε. PB obtains an assignment solution that indicates a node map between graphs. Then, BP takes the induced edge map into account to compute the approximated graph edit distance. The LSAP has a fair number of practical cubic time algorithms (Burkard, Dell'Amico, & Martello, 2009).

The Hausdorff metric is adapted to graph context in Fischer, Plamondon, Savaria, Riesen, and Bunke (2014) considering the graph nodes as individual components. The *Hausdorff edit distance (HED)* employs the matching in a quadratic time rather than the cubic time

of BP. The algorithm treats the graph nodes as a set of elements; then it finds a mapping from each set to another one concerning Hausdorff metric. Finally, the cost of node mapping in addition to the local edge structures constitute the HED.

3.4 Graph Matching Applications in Pattern Recognition

Graph-based representation of patterns has been an intrinsic approach to describe the patterns (Conte et al., 2004). Graphs are more suitable than vectorial representation for the applications where structures are essential to identify the patterns. The recent development of graph matching algorithms motivates researchers to apply graphs in to broader domains of pattern recognition (Foggia et al., 2014).

Graphs have been used in the field of bioinformatics (Mahé, Ueda, Akutsu, Perret, & Vert, 2005) to analysis the molecular structures. In Borgwardt et al. (2005) graphs have been used to predict the protein function. Chemical properties of molecules are classified in Ralaivola, Swamidass, Saigo, and Baldi (2005) using graph representations. In web content mining approaches graph matching is used in Schenker, Bunke, Last, and Kandel (2005) and Schenker, Last, Bunke, and Kandel (2004).

In handwritten document domain, graphs based methods for recognizing characters are investigated in Suganthan and Yan (1998). The handwritings are represented by strokes as graph nodes and vectorial attributes as node relations. In Rocha and Pavlidis (1994), the authors consider graph-based prototypes of character. The candidate shapes then mapped to the prototypes with a defined set of transformations.

The BP algorithm encouraged development of graph-based keyword spotting systems by providing a faster algorithm. Considering graphemes as graph nodes, Riba et al. (2015)

proposed a keyword spotting method that constructs handwriting graphs by connecting the convex groups with edges. The approach first extracts the strokes of handwriting from convex groups of the skeleton where a complete text line is taken to account for grapheme graph. The strokes correspond to the graph nodes and connect to the adjacent ones with edges. The stroke's codebooks are obtained by *blurred shape model (BSH)* clustering algorithm (Escalera et al., 2009). Then it is used to generate the node labels. Each stroke, with the k-mean algorithm, is assigned to the corresponding codebook as a node label. The edges also are labeled with three attributes: the number of points connecting the nodes, angle, and length. The coarse-to-fine approach first identifies the potential subgraphs then matches them to the query.

Using similar paradigm, Bui et al. (2015) had developed an interactive approach to automatically extract the writing pieces. The user then can compose a graph-based keyword query using invariants. The invariants are the different shapes that a stroke can have. Based on the similarities, the invariants represent the strokes. The invariants then are labeled to the graph nodes represented by strokes. The edge connectivity of nodes is defined for the strokes on basis of being in the same connected components.

The keyword spotting system in P. Wang et al. (2014b) used the skeleton of connected components attributed to the shape context representing the handwriting. The keypoints on the skeleton correspond to the nodes of graphs. Edge connectivity is placed where the keypoints are connected with the strokes. A word may consist of several connected components that yield a collection of graphs representing a word. First, the keyword graph(s) are matched with the PB algorithm. The keywords are then retrieved by DTW alignments and PB matching. In a similar manner P. Wang et al. (2014a) applied a two-stage coarse-to-fine approach that first filters the region of interest for keywords then applies the GED based graph matching.

Furthermore, the PB algorithm proposed in Riesen, Brodić, Milivojević, and Maluckov

(2014) is used for matching skeleton based graphs representing the word images. The keypoints of word skeleton are used as graph nodes where the edges are drawn from connectivity on the images. Finally, the skeleton based graphs of word images (Fischer, Suen, Frinken, Riesen, & Bunke, 2013), are used to extract the similarity feature vector from prototypes. The embedding process becomes faster with new HED matching. Using vector space embedding, the statistical *hidden Markov models (HMM)* recognizer performs the word classification.

3.5 Graph Edit Distance

Employing graph matching algorithms require several issues to be considered with care. The exact graph matching paradigm, employing graph isomorphism is too restrictive that even a small variation in structures causes rejection. Moreover, subgraph isomorphism, being less restrictive, still requires significant substructures of the graph to have the same topology and labels to score high similarity value. Thus this paradigm is not applicable in real word graph matching applications in general.

The error tolerant paradigms, on the other hand, can quantify the dissimilarity using *inexact graph matching*. In a string of characters, the problem is addressed by the string edit distance, also known as *Levenshtein* distance to compare sequences of strings. The string edit distance inspires the idea of graph matching with a set of operations. The Levenshtein algorithm minimizes the matching cost concerning the three fundamental operations: *insertion*, *deletion*, and *substitution* of characters. The algorithm determines the distance between two strings taking the minimum number of operations into account. The dynamic programming algorithm finds the optimal alignment by computing the alignment distance matrix in quadratic time. The graph edit distance algorithm tackles

graph matching intuitively similar to the string edit distance by likewise edit operation insertion, deletion, and substitution.

Graph edit distance, as an error-tolerant graph matching, alters one graph to another with a minimum cost concerning the defined operations. The minimum cost is then used for measuring the distance between two graphs. The approach evaluates every possible mapping between graph g_1 and g_2, to obtain the proper matching with the minimum cost.

String edit distance computes the optimal alignment of the given sequences in quadratic time. The graph edit distance approach employs the tree search approach. The algorithm is *NP-hard* as it spans over the induced search tree with exponential size concerning the number of graph nodes. The A^* based algorithm employs a *heuristic* functions besides it assures the solution with the optimized cost (Hart, Nilsson, & Raphael, 1968). The algorithm remains *NP-hard* nonetheless the heuristic function employed. Accordingly, we demonstrate why it is not practical to use it despite the optimal solution.

The graph edit distance algorithm operates on arbitrary graphs without any restriction on the topology of the graph or the attributes of nodes and edges. Accordingly, the algorithm defines the dissimilarity measure by employing three fundamental operations: *substitution*, *insertion*, *deletion*. The edit operations reflect the distortion of mapping individual nodes or edges. The algorithm then calculates the optimal graph edit distance with the minimum cost concerning the defined operations.

For graphs $g_1 = (V_1, E_1, \delta_1, \nu_1)$ and $g_2 = (V_2, E_2, \delta_2, \nu_2)$ the edit distance transforms g_1 to g_2 by a sequence of edit operations. It substitutes nodes and edges of g_1 with nodes and edges of g_2 by relabeling the attributes if they vary. For the nodes and edges with the more significant deterioration, the algorithm deletes the corresponding subgraphs from g_1 and inserts the subgraphs of g_2. The sequence of operations that transform g_1 to g_2 is called the edit path E from g_1 to g_2.

The edit path $\lambda = \{e_1, ..., e_k\}$ consists a set of edit operations e_i. The substitution $e = (u \rightarrow v)$ transforms a node from the first graph $u \in V_1$ to a node in $v \in V_2$ in second graph. The deletion $(u \rightarrow \varepsilon)$ removes the node $(u \in V_1)$ where by convention mapping u to the null symbol ε represents deletions. The insertion of the node $(\varepsilon \rightarrow v)$ includes the node $(v \in V_2)$ to the edit path .

Deleting every node and edges of g_1 and inserting the entire nodes and edges of g_2 is a trivial case of graph edit distance. The trivial case is valid however it does not specify any definitive information about the proximity of graphs. The substitutions in the edit path constitute the proximity as nodes and edges have direct counterparts, i.e., the substituted nodes are comparable with minor divergence. The insertion and deletion, on the other hand, represent a noticeable difference since substitution cost would be higher than individually deleting and inserting nodes and edges.

An edit path λ represents a sequence of edit operations that transforms g_1 to g_2, however, several other edit paths can perform such transformation. The graph edit distance algorithm examines the entire set of edit paths $\Upsilon(g_1, g_2)$ to find the optimal graph matching. Nonetheless, the optimal graph edit distance relies on the cost function. The insertion and deletion operations as mentioned earlier represent the dissimilarity, however, they do not provide the proximity information. On the other hand, substitution measures proximity that shows quantitative correlation by analyzing the labels of substituted subgraphs. The cost function according to circumstances must assign a higher cost to the insertion and deletion since they state a substantial divergence. Thus the optimal edit path $\lambda_j \in \Upsilon$ has the minimum cost by balancing the number of substitutions with insertions and deletions.

The cost of individual edit operations $C(e_i), 1 \leq i \leq k$ constitutes to the cost of edit path $\lambda_j = \{e_1, ..., e_k\}$. The edit path cost $C(\lambda_j)$ given in Eq. 3.2 is therefore sum of individual edit paths $C(e_i), 1 \leq i \leq k$.

$$C(\lambda_j) = \sum_{e_i \in \lambda} C(e_i) \tag{3.2}$$

The graph edit distance then is formalized by minimizing the edit path cost. The graph edit distance corresponds to the edit path with the minimum cost in Υ formally defined in Definition 4.

Definition 4 *(Graph Edit Distance GED) The graph edit distance transforms graph g_1 to g_2 by sequence of edit operations, i.e., edit path $\lambda = \{e_1, ..., e_k\}$. The edit path λ has the minimum distance among, $\Upsilon(g_1, g_2)$, set of all edit paths between g_1 and g_2. Concerning the cost function C, the GED calculates the minimum distance as directed in Eq. 3.3.*

$$d_{GED}(g_1, g_2, C) = \min_{\lambda \in \Upsilon(g_1, g_2)} C(\lambda) \tag{3.3}$$

3.5.1 Cost Functions

The graph edit distance Eq. 3.3 is parameterized by the cost function C. This fact supports using the graph matching framework in versatile applications. One can adjust the cost function for a particular domain with the prior knowledge such as alphabets of nodes and edges labels. However, insufficient knowledge could be a drawback to have a suitable proximity measure. For example when the graphs are used to distinguish the handwriting, yet a more comprehensive study of similar and dissimilar patterns are required rather than the information about graph labels. Thus having a proper cost function requires a crucial task for achieving the desired results.

Having a valid edit path E, we can make an arbitrary edit path by inserting and deleting

an element to obtain another valid yet trivial edit path, i.e., $(u \to \varepsilon)$, $(\varepsilon \to u)$. With nonspecific arbitrary cost function C, we can repeat the situation to propagate an edit path of infinite length. In addition to the infinite length, the number of such edit paths can rise dramatically. Accordingly, we declare few weak conditions on cost function to limit the graph edit distance search to a finite set of edit paths.

The graph edit distance does not compute a metric measure generally. Not being a metric measure, a cost function can result in non-symmetric graph edit distance $d(g_1, g_2) \neq d(g_2, g_1)$. However, in the context of this thesis the graphs are compared independent of their order hence $d(g_1, g_2) = d(g_2, g_1)$ must hold true. If the cost function C satisfies the metric properties the graph edit distance can successfully examine a measurable quantity of edit paths. The mathematical definition of a distance, more precisely metric principles, must be fulfilled by the cost function to yield a proper result by graph edit distance algorithm. A valid metric distance holds four principles of *non negatively*, *triangle inequality*, *symmetry*, and *identity of indiscernible*. Therefore the properties in Eq. 3.4-3.8 assert particular conditions on the edit operations e_i costs to prevent the addressed problems.

$$C(e_i) \geq 0 \qquad \text{Non negativity for substitution} \tag{3.4}$$

$$C(e_i) > 0 \qquad \text{Positive cost for insertion and deletion} \tag{3.5}$$

$$C(X \to Y) \leq C(X \to Z) + C(Z \to Y) \qquad \text{Triangle inequality} \tag{3.6}$$

$$C(X \to Y) = C(Y \to X) \qquad \text{Symmetric property} \tag{3.7}$$

$$C(X \to X) = 0 \qquad \text{Identical substitution cost} \tag{3.8}$$

The edit operations e_i are parameters of cost function C where $X \to Y$ emphasize mapping X to Y. The substitution operations for nodes or edges of identical labels can have zero

cost provided by *non negativity* principle. The insertion and deletion operations, however, do not represent similarity hence they should have a positive value. A series of insertions and deletions with the cost of zero do not change the optimal GED value, yet it could create an infinite number of edit paths. Therefore, insertion and deletion operations must have nonzero property to prevent GED from the series of augmented insertion and deletion with zero cost.

The *triangle inequality* as a property of metric distance ensures the edit paths are not excessively long. Providing that the cost of operations $(X \to Y)$ becomes higher than $(X \to Z)$ and $(Z \to Y)$, GED takes the low cost yet longer path into account. When *triangle inequality* conditions hold true on cost function the right side of inequality always has a higher value than the left side. Thus the edit paths maintain the smaller number of operations and lower matching cost eventually. Intuitively for assigning nodes, a direct assignment is preferred over a path with intermediate nodes.

The *symmetry* property preserves the equality of an edit operation with its inverse. Comparing two substructures is independent of the order, for example, substituting $(u \to v)$ has the same cost as of replacing $(v \to u)$. As a consequence of violating the symmetry requirement, the graph edit distance of g_1 and g_2 can yield different values, i.e. $d(g_1, g_2) \neq d(g_2, g_1)$.

Finally, the cost function must satisfy *identity of indiscernible*. Considering the identical attributes of objects the cost as a result is zero. The principle in graph matching context states substituting a node or edge with itself has a zero cost. We would refer the reader to Bunke and Allermann (1983) for more information on the metric.

The functions in 3.9 and 3.10 define the Euclidean cost function for vectorial labels. These functions fulfill the valid metric distance properties Eq. 3.4-3.8. The Euclidean cost function assists the graph edit distance algorithm to succeed and acquire a proper result.

$$\text{Node costs} \begin{cases} C(u \to v) = \alpha \cdot ||\delta_1(u) - \delta_2(v)|| & \text{Substitution} \\ C(u \to \varepsilon) = \alpha \cdot \tau_n & \text{Deletion} \\ C(\varepsilon \to v) = \alpha \cdot \tau_n & \text{Insertion} \end{cases} \tag{3.9}$$

$$\text{Edge costs} \begin{cases} C(p \to q) = (1-\alpha).||\nu_1(p) - \nu_2(q)|| & \text{Substitution} \\ C(p \to \varepsilon) = (1-\alpha) \cdot \tau_e & \text{Deletion} \\ C(\varepsilon \to q) = (1-\alpha) \cdot \tau_e & \text{Insertion} \end{cases} \tag{3.10}$$

The *insertion/deletion* operations have constant costs of $\tau_n > 0, \tau_e > 0$ as the costs of nodes and edges respectively. As a reminder to abide the symmetry requirement an identical value for insertion and deletion has been chosen. The shared parameter between nodes and edges cost function α is weighting parameter. Being $0 \leq \alpha \leq 1$, the parameter defines the priority of nodes over edges in the computation of graph edit distance. For instance, setting $\alpha = 0.5$ balances the equal contributions between nodes and edge.

A substitution has two corresponding nodes or edges. The Euclidean cost function for substitution operation measures the distance of corresponding labels. The far away labels highlight a considerable difference between two ends. Therefore they are associated with a higher cost. A node or edge substitution can attain at most $2 \times \tau_n$ and $2 \times \tau_e$ respectively otherwise they will be replaced by a deletion and insertion of corresponding nodes or edges.

3.5.2 A^* Algorithm for Graph Edit Distance

The graph edit distance is fundamentally an A^* *best-first search* algorithm (Hart et al., 1968). The A^* search, Algorithm 2, conducts a tree search by creating an ordered tree. The tree comprises all feasible edit paths that map the nodes and edges of g_1 to g_2. The leaves serve as the complete edit path starting from the root of the tree. The optimal graph edit distance is, however, a complete edit path with the minimum cost. Furthermore, a path from root to an inner node of the tree denotes a partial edit path. The A^* search tree does not create the entire tree at the beginning; it slightly expands the partial solutions concurrently with the search. An inner node p in the ordered tree represents a partial solution from tree root ϕ to p. The cost component $g(p)$ represents the determined cost of path from root ϕ to p based on the included node maps. However, the heuristic function $h(p)$ predicts the cost of upcoming edit operation from p to a leaf node to yield a complete edit path. The algorithm obtains a cost information $f(p) = g(p) + h(p)$ for every partial solution. Afterwards, based on the cost information $f(p)$, the algorithm expands the path with the minimum cost. Eventually, the algorithm discovers a complete edit path which is anticipated to be the optimal edit map as well. The algorithm stops at the first complete edit path it approaches since the algorithm performs the best-first search. Figure 3.2 shows the entire search tree for graphs with $V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{u_1, u_2\}$ as set of nodes. ε represents deletion $(u \to \varepsilon)$ or insertion $(\varepsilon \to v)$ respectively.

The search through the tree paths examines every possible nodes map $E \in \gamma(g_1, g_2)$ between two graphs g_1 and g_2. A node map $E = \{e_1, ..., e_{|E|}\}$ consists a set of edit operations e_i which are insertion, deletion or substitution of nodes. Each node $u \in g_1$ and $v \in g_2$ must appear at most once in a valid node map, in other words, multiple assignments of a node are not permitted. Accordingly, a node map is partial unless every node of g_1 and g_2 appears in the node map. The GED algorithm 2 finds the optimal complete node map $E = \{e_1, ..., e_{|M|}\}$ by exploring all the possible node maps in $E \in \gamma(g_1, g_2)$.

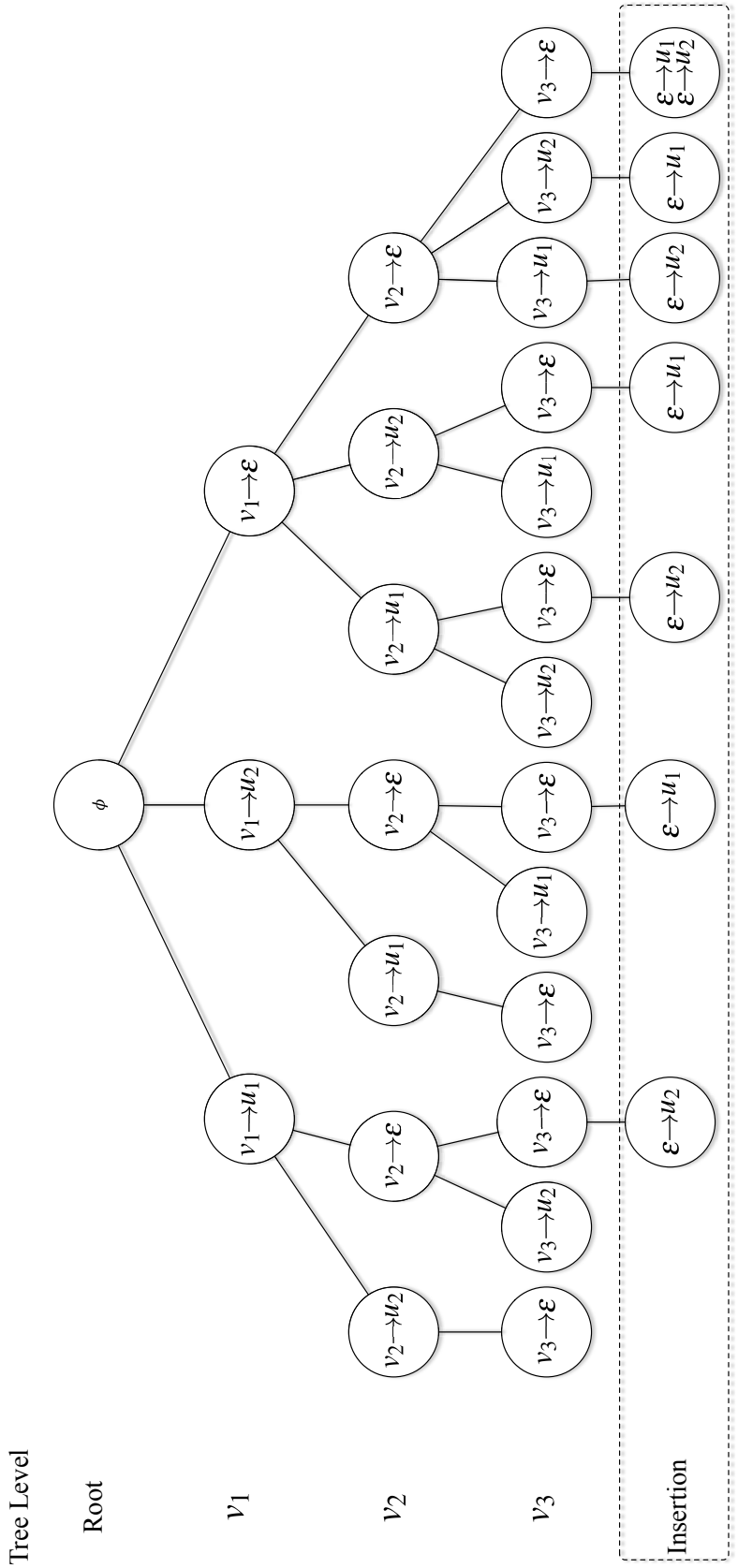


FIGURE 3.2: GED search tree for graph with $V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{u_1, u_2\}$

Algorithm 2 Graph Edit Distance Algorithm

Require: Graphs $g_1 = (V_1, E_1, \delta_1, \nu_1)$ and $g_2 = (V_2, E_2, \delta_2, \nu_2)$,
Where $V_1 = \{u_1, ..., u_{|V_1|}\}$ and $V_2 = \{v_1, ..., v_{|V_2|}\}$.
Ensure: A minimum cost edit path, $p_{\min}$, from g_1 to g_2.
$OPEN \leftarrow \phi$
for all $v \in V_2$ **do**
    $OPEN \leftarrow \{(u_1 \rightarrow v)\}$
end for
$OPEN \leftarrow \{(u_1 \rightarrow \varepsilon)\}$
loop
    $p_{\min} = \arg\min_{p \in OPEN}\{g(p) + h(p)\}$
    $OPEN \leftarrow OPEN \setminus p_{\min}$
    if $p_{\min}$ is a complete edit path **then**
        Return $p_{\min}$ ▷ Solution
    else
        Let $p_{\min} = \{(u_1 \rightarrow v_{i1}), ..., (u_k \rightarrow v_{ik})\}$
        if $k < |V_1|$ **then**
            for all $v \in V_2 \setminus \{v_{i1}, ..., v_{ik}\}$ **do**
                $OPEN \leftarrow OPEN \cup \{p_{\min} \cup \{(u_{k+1} \rightarrow v)\}\}$
            end for
            $OPEN \leftarrow OPEN \cup \{p_{\min} \cup \{(u_{k+1} \rightarrow \varepsilon)\}\}$
        else
            $OPEN \leftarrow OPEN \cup \{p_{\min} \cup \{(\varepsilon \rightarrow v_{ik+1}), \ldots, (\varepsilon \rightarrow v_{i|V_2|})\}\}$
        end if
    end if
end loop

Algorithm 3 computes the edit path cost $g(p)$. The edit path consists of the nodes and induced edges operations. A complete node map is represented with M. The edit path cost function employs node cost 3.9 and edge cost 3.10 on the induced edit path to calculate the costs.

Algorithm 3 EPC(M,C)

Require: complete node map M, cost function C
Ensure: edit path cost c

```
1:  c ← 0
2:  for all node deletion (u → ε) in M do
3:      c ← c + C(u → ε)
4:      for all implied edge deletion (p → ε) do
5:          c ← c + C(p → ε)/2
6:      end for
7:  end for
8:  for all node insertion (ε → v) in M do
9:      c ← c + C(ε → v)
10:     for all implied edge insertion (ε → q) do
11:         c ← c + C(ε → q)/2
12:     end for
13: end for
14: for all node substitution (u → v) in M do
15:     c ← c + C(u → v)
16:     for all implied edge substitution (p → q) do
17:         c ← c + C(p → q)/2
18:     end for
19:     for all implied edge insertion (ε → q) do
20:         c ← c + C(ε → q)/2
21:     end for
22:     for all implied edge deletion (p → ε) do
23:         c ← c + C(p → ε)/2
24:     end for
25: end for
26: return c
```

The root of the tree ϕ, i.e., the beginning point, contains no node map. On each subsequent level of the tree, GED opens the path including a node of the first graph $v \in V_1$. More precisely, the operations comprise the deletion $v \to \varepsilon$ and substitution $v \to u_i$ of v with every unassigned node $u_i \in V_2$. For instance, the search tree dedicates the first level of

the tree to the $v_1 \in V_1$. The edit operations $v_1 \to \varepsilon$, and $v_1 \to u_i$ constitute the deletion and substitution of v_1 respectively where $u_i \in V_2$. Therefore GED opens $|V_2| + 1$ unique paths in the first level of tree search.

For every subsequent tree level $j = 2, ..., |V_1|$, a similar situation will take place. However, the cost function $f(p) = g(p) + h(p)$ determines the forthcoming node p. Nonetheless, the search expands a partial path on the tree independent of the tree level after that. In addition to the deletion of node $(v_j \to \varepsilon)$, the expansion of the tree substitutes $(v_j \to u_i)$ for $u_i \in V_2$ concerning u_i has not appeared in the induced node map.

Eventually, the level $j = |V_1|$ processes the last node v_j by either deleting or substituting. For any partial edit map at this level, the tree can expand just one level to $j = |V_1| + 1$. However, for the nodes from V_2 have not appeared, being at $j = |V_1| + 1$, no substitution can take place since all source graph nodes $v \in V_1$ are already consumed. Consequently, the remaining nodes $u_i \in V_2$ in line 20 of algorithm 2, are inserted $(\varepsilon \to u_i)$ at once to make a complete map.

The graph edit distance comprises nodes and edge edit operations. The edge maps are under the restriction of the underlying node maps. Hence, If a node is deleted $(v \to \varepsilon)$ or inserted $(\varepsilon \to u)$ in the node maps, the corresponding adjacent edges will be removed or added respectively. For example, when the node $(v \to \varepsilon)$ is removed, the every adjacent edge q will be removed accordingly $(q \to \varepsilon)$.

For induced edge substitution, an edge $r \in E_1$ could be matched to another edge $q \in E_2$ only when nodes are substituted on both ends. More precisely, edge substitution $(r \to q)$ is induced when the ending points of $r = (v, v\prime)$ are matched to ending points of $q = (u, u\prime)$. Hence the node map must contain $(u \to v)$ and $(u\prime \to v\prime)$. In any other circumstances, the edges r, q would be deleted $(r \to \varepsilon)$, and inserted $(\varepsilon \to q)$ respectively.

Having $m = |V_1|$ and $n = |V_2|$ number of nodes, the maximum number of node substitutions in the node map, as substitution requires a corresponding node, is $\min(m,n)$. The remaining nodes map then contain either $m+n-2*\min(m,n)$ deletions or insertions. Therefore, the edit path p with minimum number of operations has size of $|p| = m+n-\min(m,n)$ node operations in total. On the other hand, the maximum number of operations arises when there is no substitution involved in edit path which happens for notably different graphs. The GED represents this situation, through removing every node from the first graph by m operations and inserting every node from the second graph by n insertions. The edit path with a maximum number of edit operations hence has a size of $|p| = m+n$. Consequently, the tree search algorithm traverses at least m levels of tree regarding the V_1 nodes. The tree expands by a proportional factor to n at each level. Accordingly, the graph edit distance is applicable in small problems yet by increasing graphs size it is not feasible to approach in practical applications.

For a partial edit path, the function $g(p)$ determines the cost of the induced path. The A^* search, however, estimates the cost of remaining operations by a heuristic function. The heuristic function $h(p)$, on the other hand, approximates the cost of the unmapped nodes. The heuristic cost, however, must be *admissible*, the predicted value must be lower or equal to the actual cost. Reaching the first complete map in *OPEN* is guaranteed to be the optimal solution with an appropriate heuristic cost that does not overestimate the real values. Hence, by merely setting heuristic to zero, $h(p) = 0$, we fulfill this requirement; however, a useful heuristic can help to reduce the tree size and computation.

3.6 Hausdorff Edit Distance

The GED obtains an optimal solution, yet it has an exponential time complexity. Due to the time complexity, it is unlikely to use GED for the graphs of medium to large size. The

alternative graph matching, with a weaker constraint on the edit path, can be achieved by approximating the GED. Conventional approaches for the approximation of GED consider the *bipartite graphs* as graph matching. Hence the approximation approach relaxes the global constraint to a local matching of nodes and their local structures.

A bipartite graph has its set of nodes V separated in two disjoint sets of nodes V' and V''. No connection exists between nodes of the same set; accordingly, the edges connect the nodes from disjoint node sets between $u \in V'$ and $v \in V''$. Definition 5 explains the bipartite graphs more precisely.

Definition 5 *(Bipartite Graph) A bipartite graph* $g = (V, E, \delta, \nu)$ *satisfies the condition of having nodes in two partition which is* $V = V' \cup V''$ *and* $V' \cap V''$. *The edge are subset of* $E \in V' \times V'' \cup V'' \times V'$.

Considering the graph matching as a bipartite graph, we can mention the *BP* (Riesen, Fischer, & Bunke, 2015) and *Hausdorff Edit Distance (HED)* (Fischer et al., 2014, 2015) with cubic and quadratic time complexity respectively. These algorithms approximate the graph edit distance employing *bipartite graph* concept.

The bipartite graph matching for g_1 and g_2 considers V_1 and V_2 as two node partitions $V\prime$ and $V\prime\prime$. The algorithm, supporting bipartite constraints in Definition 5, associates a mapping between nodes in $V\prime$ and $V\prime\prime$. The resulting bipartite graph serves as an edit path between g_1 and g_2. The bipartite mapping, compared to GED, relaxes the global constraint on edit map to local subgraph structures. Therefore, the local adjacency of edges is taken into account considering individual nodes in the edit path.

Figure 3.3 shows an example of bipartite graph that is separated in two sets $V_1 = \{u_1, u_2\}$ and $V_2 = \{v_1, v_2, v_3\}$. The edges relate the nodes of disjoints sets. The adjacency denotes a valid edit path if it associates each node with precisely one node. Concerning insertion or deletion, helper node(s) ε in V_1 and V_2 are enough to show these operations.

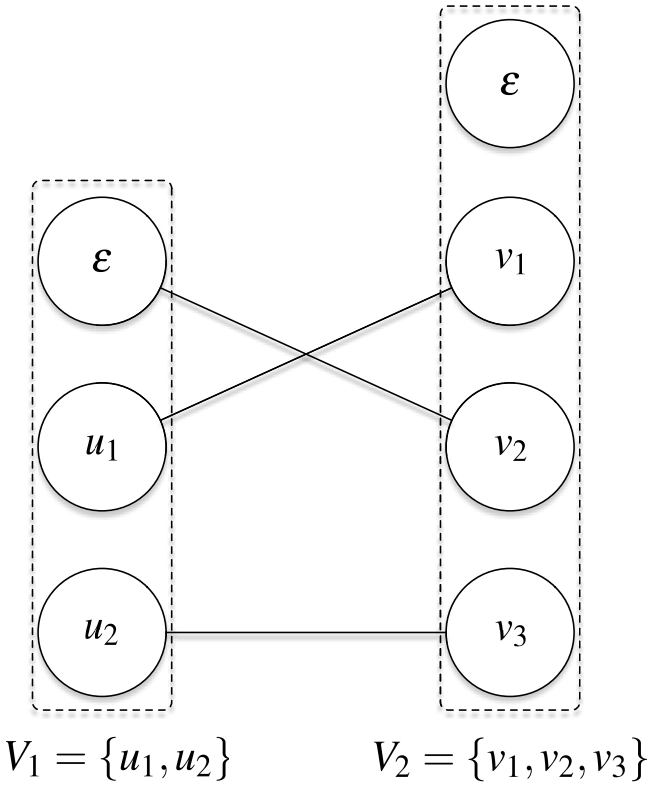


FIGURE 3.3: A bipartite graph maps V_1 and V_2.

Hence, in the example the implied edit path contains three node maps: two substitutions $(u_1 \to v_1), (u_2 \to v_3)$, and one insertion $(\varepsilon \to v_2)$.

In a metric space, a distance exists between every two members. In two independent subsets of a metric space the concept of distance between two sets can be inferred from distance of individual members. For instance, we can use the *minimin* in Eq. 3.11 to find the distance of nearest members between two sets.

$$\text{minimin}(A,B) = \min\left\{ \inf_{a\in A}\left(\inf_{b\in B}\left(d(a,b)\right)\right), \inf_{b\in A}\left(\inf_{a\in A}\left(d(a,b)\right)\right)\right\} \tag{3.11}$$

However, by simply taking the distance of two close points as distance we exclude the underlying information about other members in the set. Figure 3.4 shows the nearest

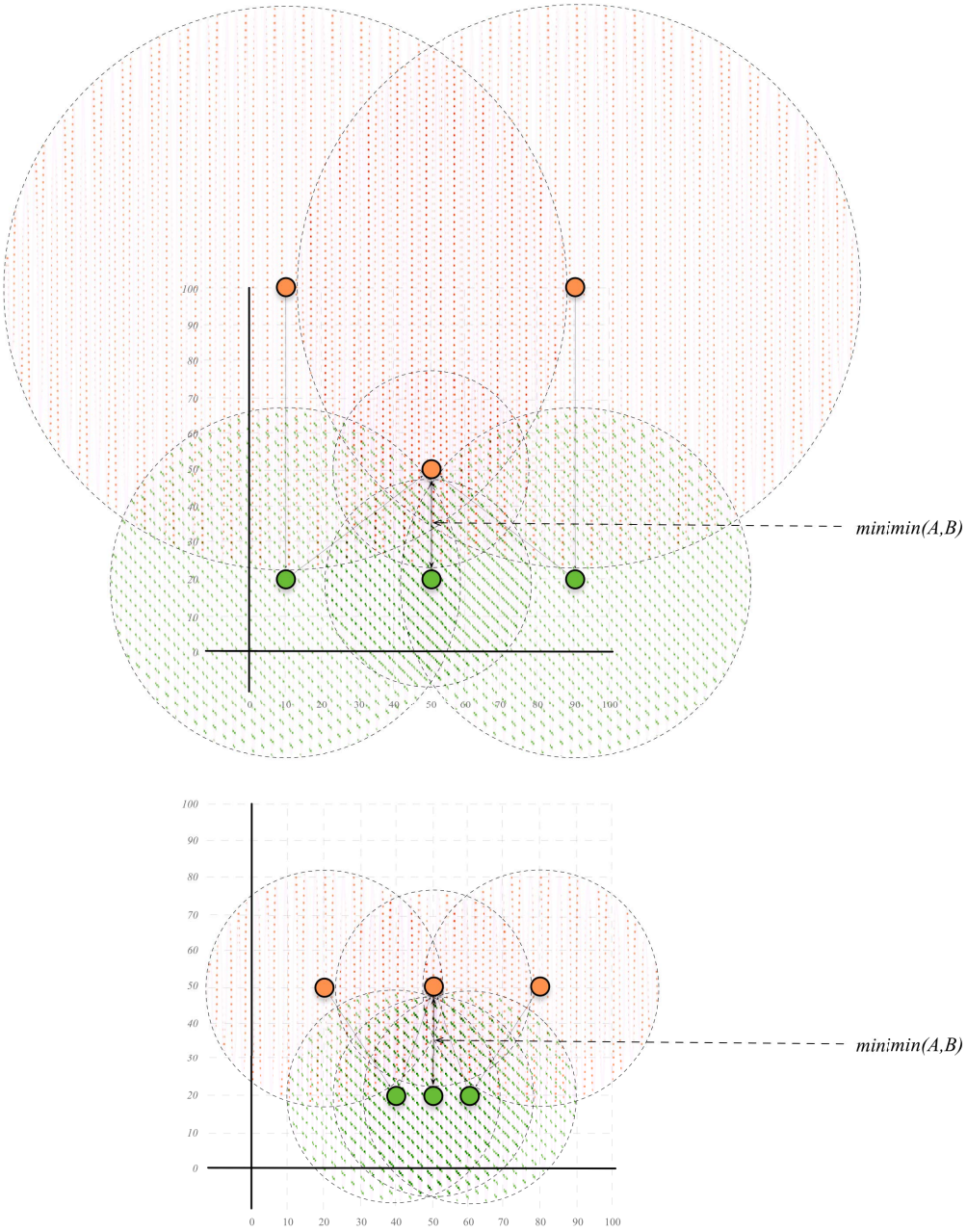


FIGURE 3.4: Nearest distance by means of *minimin* function between sets *A* and *B* represented with orange and green colors respectively.

distance of two sets of points. The nearest distances in two examples are equal when *minimin* function is employed. However, we can observe a considerable difference in the given cases. The *minimin* distance does not delegate a suitable measure between two sets of points as it does not consider the entire members.

The Hausdorff[1] metric, names after Felix Hausdorff[2], is a *maximin* metric that calculates the distance between two sets. Having sets A and B, the oriented Hausdorff distances are obtained by:

$$h(A,B) = \sup_{a \in A} \left(\inf_{b \in B} \left(d(a,b) \right) \right) \tag{3.12}$$

$$h(B,A) = \sup_{b \in A} \left(\inf_{a \in A} \left(d(a,b) \right) \right). \tag{3.13}$$

The distances $h(A,B)$ and $h(B,A)$, being *asymmetric*, are not necessarily equal. The more general Hausdorff distance is defined in Eq. 3.14 by taking the maximum of $h(A,B)$ and $h(B,A)$ distances. From now on we consider the symmetric $H(A,B)$ unless specifically mentioned.

$$H(A,B) = \max \left\{ \sup_{a \in A} \left(\inf_{b \in B} \left(d(a,b) \right) \right), \sup_{b \in A} \left(\inf_{a \in A} \left(d(a,b) \right) \right) \right\} \tag{3.14}$$

Figure 3.5 shows examples of Hausdorff distance between sets A and B. Compared to *minimin* distance of the same sets in Figure 3.4, the Hausdorff distance can make a better metric to distinguish the differences.

For finite sets of A and B, we adapt the Hausdorff distance in Definition 6.

[1] Also called Pompeiu–Hausdorff distance.

[2] Felix Hausdorff was a German mathematician (November 8, 1868 – January 26, 1942).

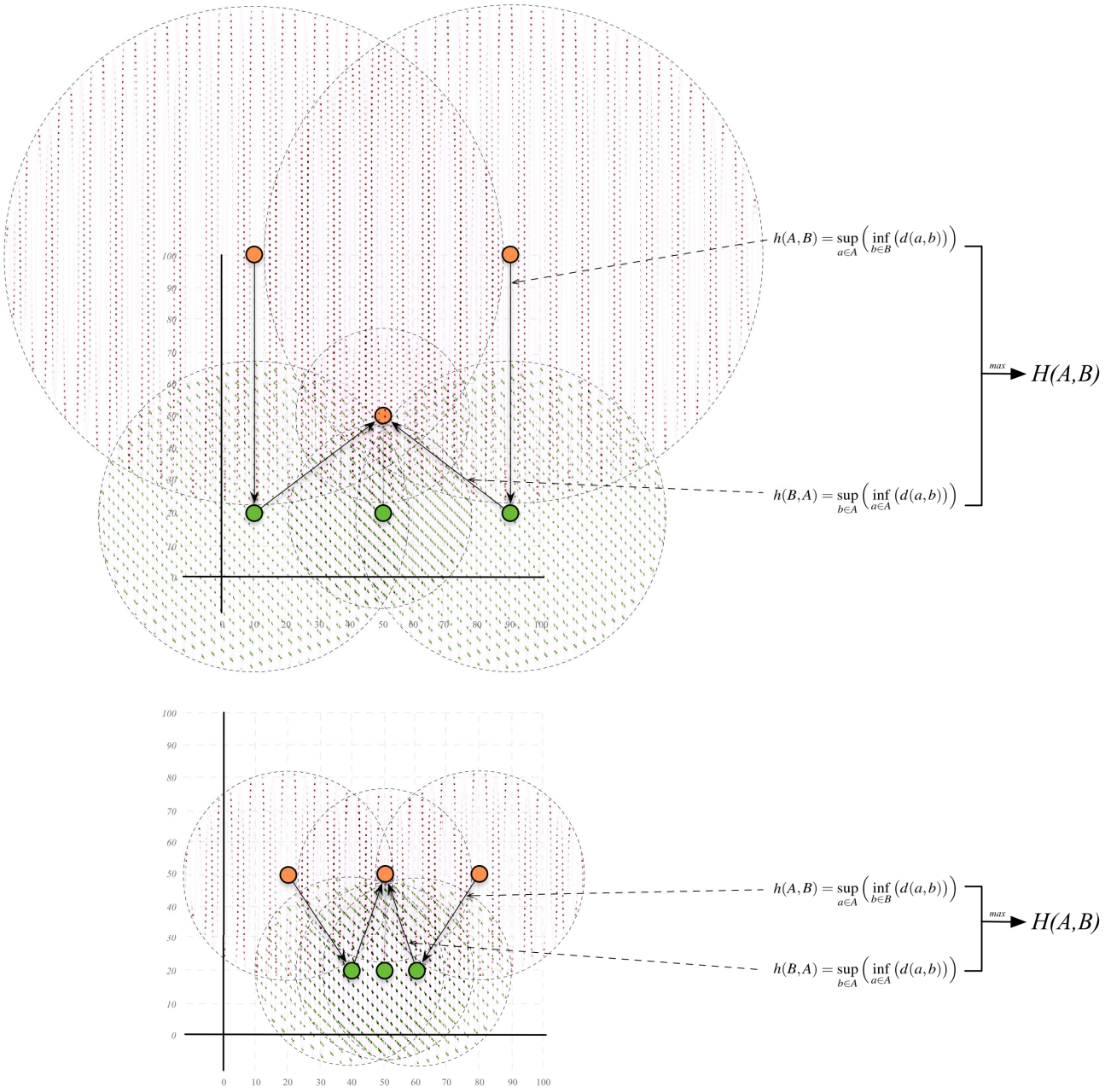


FIGURE 3.5: Hausdorff distance between sets A and B represented with orange and green colors respectively.

Definition 6 *(Hausdorff Distance) The Hausdorff distance for finite sets* $A = \{a_1, ..., a_n\}$ *and* $B = \{b_1, ..., b_m\}$ *is defined by Eq. 3.15*

$$H(A,B) = \max\left\{ \max_{a\in A}\left(\min_{b\in B}\left(d(a,b)\right)\right), \max_{b\in A}\left(\min_{a\in A}\left(d(a,b)\right)\right)\right\} \tag{3.15}$$

HED measures the distance of two graphs by means of the Hausdorff distance. The algorithm in Fischer et al. (2015) employs a modified Hausdorff function to obtain the distance. Considering possible outliers the modified metric Eq. 3.16 provides a better representation by including the entire nearest distance into the equation. The equation uses the summation instead of the maximum functions. The distance $d(a,b)$ in Eq. 3.16 can be any metric such as Euclidean distance between a and b.

$$H(A,B) = \sum\left\{ \sum_{a\in A}\left(\min_{b\in B}\left(d(a,b)\right)\right), \sum_{b\in A}\left(\min_{a\in A}\left(d(a,b)\right)\right)\right\} \tag{3.16}$$

Considering graph nodes and their local edge connectivity as two sets, HED employs the Hausdorff function on sets to estimate GED. The algorithm obtains a mapping between graph members as a result. The HED mapping, however, is directed compared to the GED edit path. Thus the node assignments may not be symmetric Eq. (3.12-3.13). A node accordingly can receive multiple assignments. Figure 3.6 shows an example of the bipartite graph with directed node maps. The node map contains $\{(u_1 \to v_1), (u_2 \to v_3), (v_1 \to \varepsilon), (v_2 \to u_2), (v_3 \to u_2)\}$. Unlike undirected node map in Figure 3.3, the HED node map is directed hence it is not unnecessarily symmetric. Also, multiple node assignments, i.e., assignment to u_2 by $(v_2 \to u_2), (v_3 \to u_2)$, is permitted by HED.

Hausdorff Edit Cost (HEC) computes the Hausdorff distance between two arbitrary sets. Figure 3.7 demonstrates the HEC algorithm for sets A and B. The algorithm 4 calculates

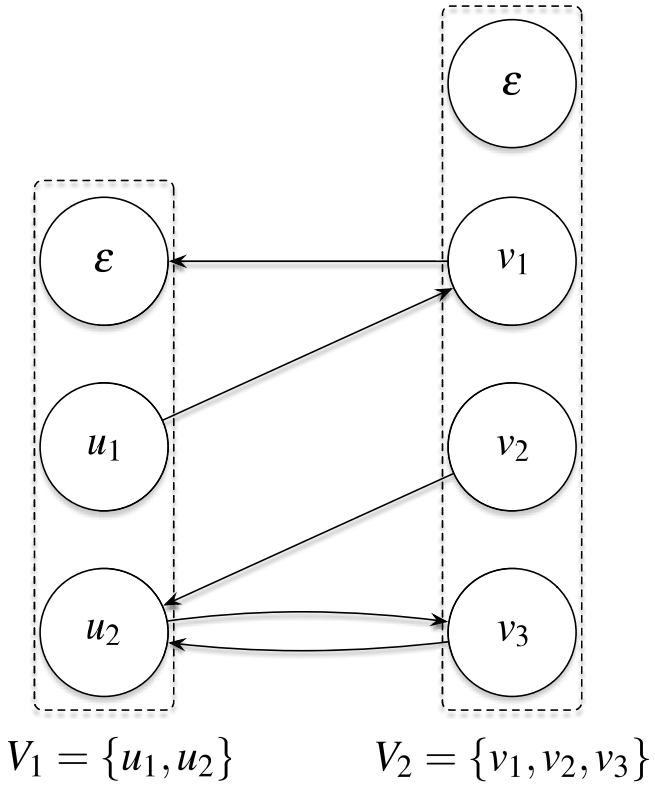


FIGURE 3.6: A directed bipartite graph represents an HED mapping.

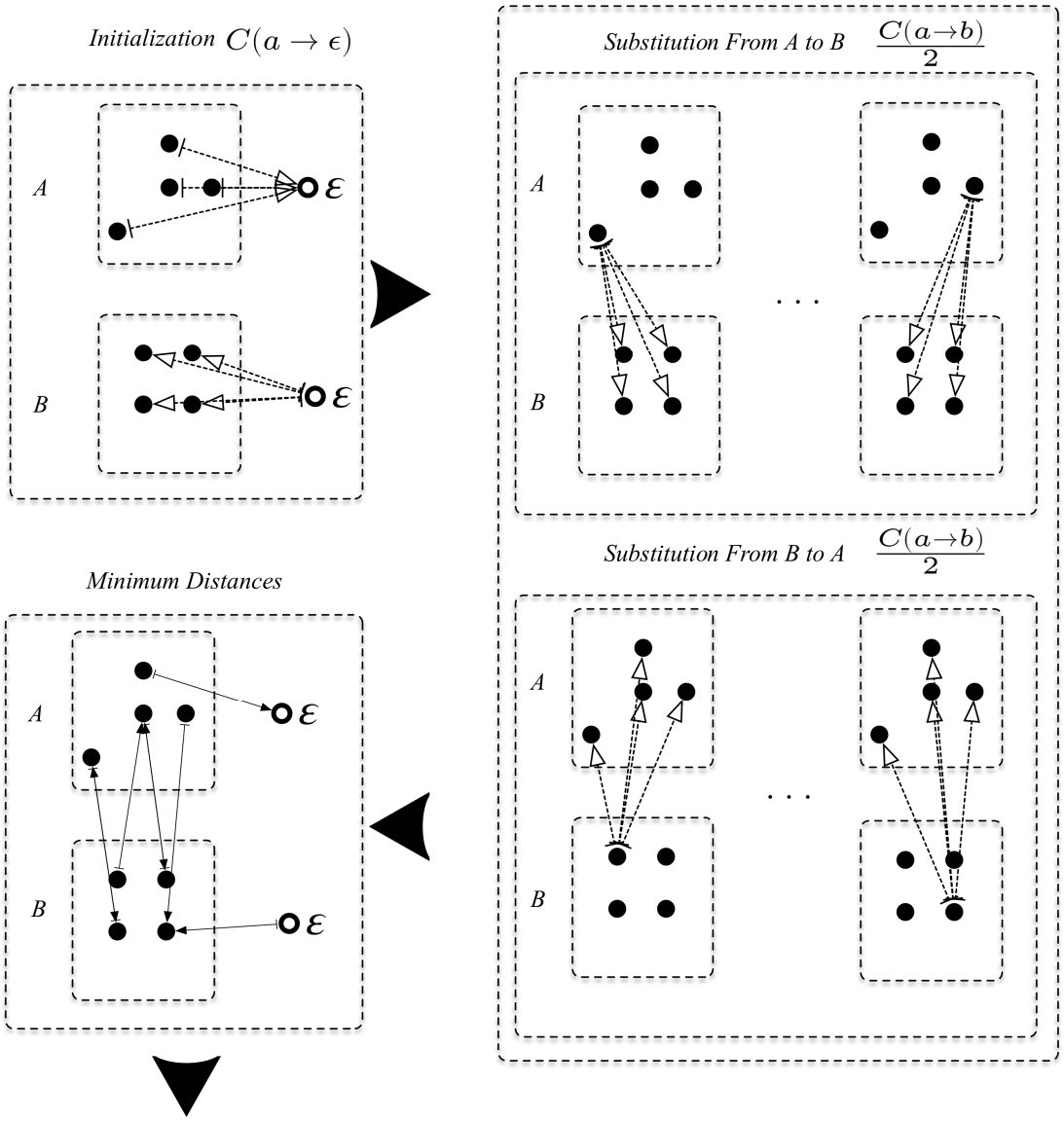
the Hausdorff distance in Eq. 3.16 for sets A and B with respect to the cost functions in Eq. 3.9-3.10.

The node cost function needs an adjustment with regard to the individual type of edit operations. Considering the cost function C in Eq. 3.9-3.10, the HED node cost is obtained by Eq. 3.17-3.19.

$$C_n^*(u,\varepsilon) = C(u \to \varepsilon) + \sum_{p_i \in u.E} \frac{C(p_i \to \varepsilon)}{2} \tag{3.17}$$

$$C_n^*(\varepsilon,v) = C(\varepsilon \to v) + \sum_{q_i \in v.E} \frac{C(\varepsilon \to q_i)}{2} \tag{3.18}$$

$$C_n^*(u,v) = \frac{C(u \to v) + \frac{C_e(u,v)}{2}}{2} \tag{3.19}$$



$$HEC \leftarrow \sum_{a \in A} c_1(a) + \sum_{b \in B} c_2(b)$$

FIGURE 3.7: HEC algorithm diagram shows computation of modified Hausdorff distance between sets *A* and *B*.

Algorithm 4 HEC(A,B,C) Hausdorff Edit Cost

Require: A, B , cost function C
Ensure: Hausdorff edit cost c
for all $a \in A$ **do**
    $c_1(a) \leftarrow C(a \rightarrow \varepsilon)$
end for
for all $b \in B$ **do**
    $c_2(b) \leftarrow C(b \rightarrow \varepsilon)$
end for
for all $a \in A$ **do**
    for all $b \in B$ **do**
        $c_1(a) \leftarrow \min\left(\frac{C(a \rightarrow b)}{2}, c_1(a)\right)$
        $c_2(b) \leftarrow \min\left(\frac{C(a \rightarrow b)}{2}, c_2(b)\right)$
    end for
end for
$c \leftarrow \sum_{a \in A} c_1(a) + \sum_{b \in B} c_2(b)$
return c

We show the set of edges adjacent to u and v with $u.E$ and $v.E$ respectively in Eq. 3.17-3.19. Three types of node matching cost are described: deletion, insertion, and substitution. In case of insertion or deletion of nodes, the cost includes cost of node insertion or deletion and half the cost of adjacent edges. The other half of edges costs depends on the node on the other side of the edges.

In the case there is a substitution (u,v) half the cost of mapping node and a quarter of adjacent edge costs C_e is taken into account. Likewise, the other half of node cost is reserved for the opposite direction.

The edge cost could be defined with a similar argument without constraint of node map as in Eq. 3.20-3.22.

$$C_e^*(p \to \varepsilon) = C(p \to \varepsilon) \tag{3.20}$$

$$C_e^*(\varepsilon \to q) = C(\varepsilon \to q) \tag{3.21}$$

$$C_e^*(p \to q) = \frac{C(p \to q)}{2} \tag{3.22}$$

The edges adjacent to u and v are represented by sets $u.E = \{p_1, ..., p_{|u|}\}$ and $v.E = \{q_1, ..., q_{|v|}\}$ respectively. Therefore, every edge in $u.E$ can be mapped to every edge in $v.E$. The Hausdorff distance of these sets $C_e(u,v)$, given in Eq. 3.23, can be computed with algorithm 4. Thus the function $HEC(u.E, v.E, C)$ computes $C_e(u,v)$ which is the edge cost related to substitution $(u \to v)$. C_e is less than the true cost because it does not consider the mapping constraint in computation.

$$C_e(u,v) = \sum_{p \in u.E} \min_{q \in v.E + \varepsilon} C^*(p \to q) + \sum_{q \in v.E} \min_{p \in u.E + \varepsilon} C^*(p \to q) \tag{3.23}$$

The lower bounds are introduced to compensate the underestimation of the costs. The $\mathrm{HED}(g_1, g_2, C)$ lower bound is derived by assuming each node can be mapped to other nodes. Eq. 3.24 calculates the lower bound assuming every node in V_1 and V_2 can be mapped with zero cost. Hence the contributing factor to the lower bound is the difference between $|V_1|$ and $|V_2|$. The remaining nodes are deleted or inserted with minimum cost.

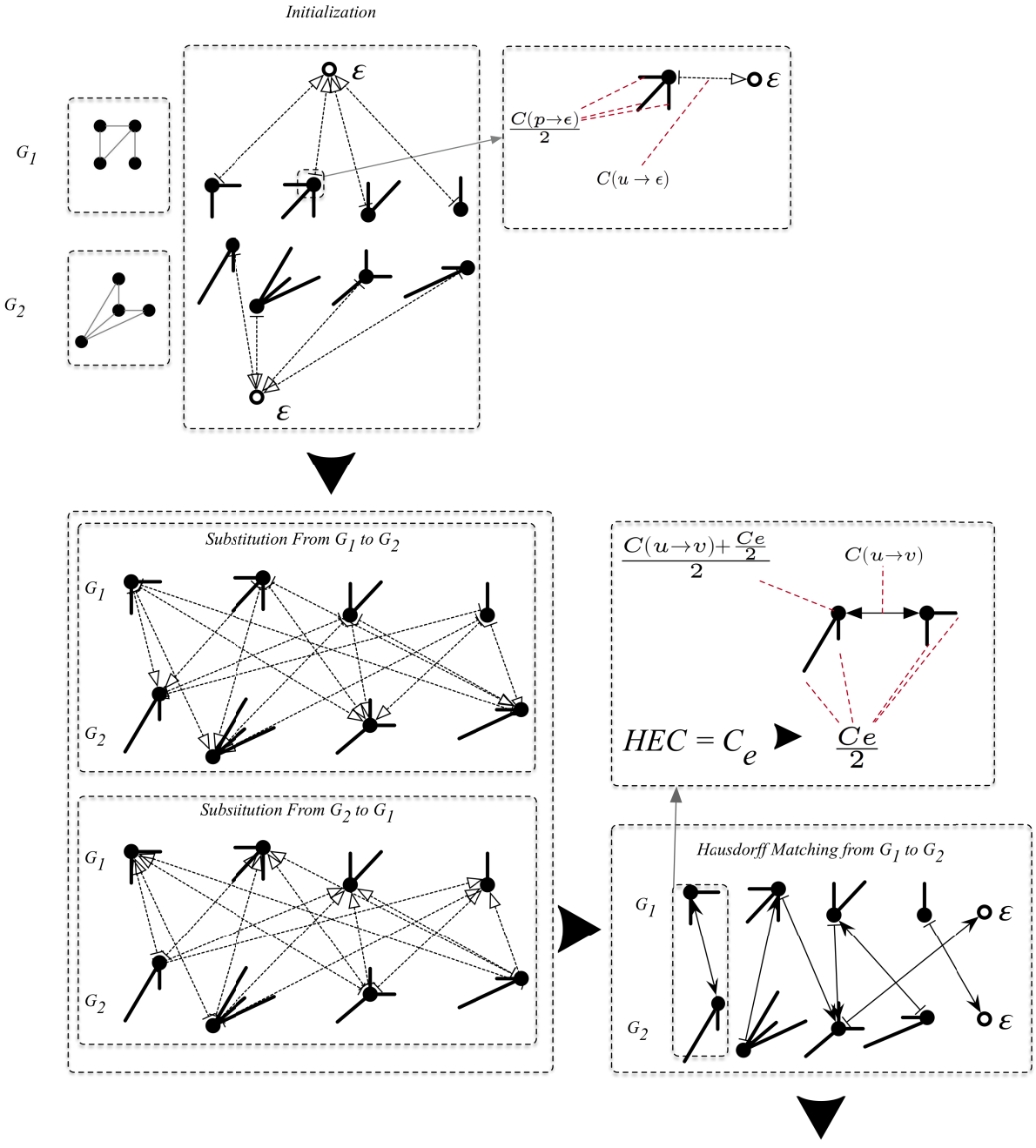
$$L(g_1, g_2) = \begin{cases} (|V_1| - |V_2|).\min_{u \in V_1} C(u \to \varepsilon) & |V1| > |V2| \\ (|V_2| - |V_1|).\min_{v \in V_2} C(\varepsilon \to v) & \text{o.w.} \end{cases} \tag{3.24}$$

With the same argument for nodes $u \in V_1$ and $v \in V_2$, the edge cost C_e lower bound is achieved by Eq. 3.25.

$$L(u,v) = \begin{cases} (|u| - |v|).\min_{p \in P} C(p \to \varepsilon) & |u| > |v| \\ (|v| - |u|).\min_{q \in Q} C(\varepsilon \to q) & \text{o.w.} \end{cases} \tag{3.25}$$

The Hausdorff edit distance computes suboptimal solution for graph edit distance. Since the node operations are considered individually the edge context consequently are evaluated locally. The local context assumes every adjacent edge can be mapped to other edges regardless of the other ends. Although the assumption might be invalid and real edge cost might be higher than the computed one. In general HED underestimates the cost computed by GED, that is $HED(g_1, g_2) \leq GED(g_1, g_2, C)$. Figure 3.8 demonstrates graph matching using local edge context by HED algorithm. The final graph matching is a directed bipartite graph.

Algorithm 5 calculates the HED(g_1, g_2, C). The algorithm initializes d_1 and d_2 by deletion and insertion edit operations. The cost includes the cost of deletion or inserting nodes and half the cost of adjacent edges. Then the substitution of nodes takes place in lines $7-14$. Comparing every two pairs of nodes $(u,v) \in g_1.V \times g_2.V$, the algorithm starts with calculating C_e at line 9 by calling HEC$(u.E, v.E, C)$ function. Next, the maximum of the lower bound $L(u,v)$ and C_e are taken to account to avoid underestimation of C_e. The node substitutions, which should be smaller than deletion or insertion, are calculated and adjusted accordingly at lines $11-12$. d_1 and d_2 constitute the potential HED that is achieved by summing up the included costs. Finally, line 16 verifies the calculated cost and would adjust it with the lower bound $L(g_1, g_2)$ if the cost is underestimated. The HED algorithm has quadratic time complexity concerning the nested loops that iterate on $g_1.V$ and $g_2.V$. More precisely, having $n_1 = |g_1.V|$ and $n_2 = |g_2.V|$ nodes the time complexity would be $O(n_1.n_2)$.



$$\text{HED} = \sum_{u \in g_1.V} d_1(u) + \sum_{v \in g_2.V} d_2(v)$$

FIGURE 3.8: HED algorithm diagram shows graph matching using Hausdorff edit distance between graphs G_1 and G_2.

Algorithm 5 HED(g_1, g_2, C) Hausdorff Edit Distance

Require: g_1, g_2 and cost function C
Ensure: HED distance d

for all $u \in g_1.V$ **do**
  $d_1(u) \leftarrow C(u \rightarrow \varepsilon) + \sum_{p \in u.E} \frac{C(p \rightarrow \varepsilon)}{2}$
end for
for all $v \in g_2.V$ **do**
  $d_2(v) \leftarrow C(\varepsilon \rightarrow v) + \sum_{q \in v.E} \frac{C(\varepsilon \rightarrow q)}{2}$
end for
for all $u \in g_1.V$ **do**
  for all $v \in g_2.V$ **do**
    $C_e \leftarrow HEC(u.E, v.E, C)$
    $C_e \leftarrow \max(L(u,v), C_e)$
    $d_1(u) \leftarrow \min\left(\frac{C(u \rightarrow v) + \frac{C_e}{2}}{2}, d_1(u)\right)$
    $d_2(v) \leftarrow \min\left(\frac{C(u \rightarrow v) + \frac{C_e}{2}}{2}, d_2(v)\right)$
  end for
end for
$d \leftarrow \sum_{u \in g_1.V} d_1(u) + \sum_{v \in g_2.V} d_2(v)$
$d \leftarrow \max(d, L(g_1, g_2))$
return d

Chapter 4

Graph-based Keyword Spotting

Graph representation is an essential part of our KWS paradigm. The representations in pattern recognition impact the succeeding steps of the system. In this chapter, we present our approach for using the graph-based representation in our proposed KWS system in chapter 5. We discuss possible representations of the word images with graphs. The graphs have attributes, which are used to calculate the distances. The Graph Edit Distance class of algorithms takes an entire word as a graph to perform the calculations.

4.1 Handwriting Graphs

The graph-based keyword spotting approach uses the graph for the representation of the word images. Figure 4.1 demonstrates the overview of the keyword spotting workflow. As a template-based approach, a query based on a word image is used to retrieve the words.

The primary issue to consider for graph-based KWS is how to represent handwritings in a graph form. The graph-based representation of handwritten words must capture the essential aspects of handwriting. Hence, the graph representation is as important as feature extraction in statistical recognition methods. In KWS approach, graphs associate

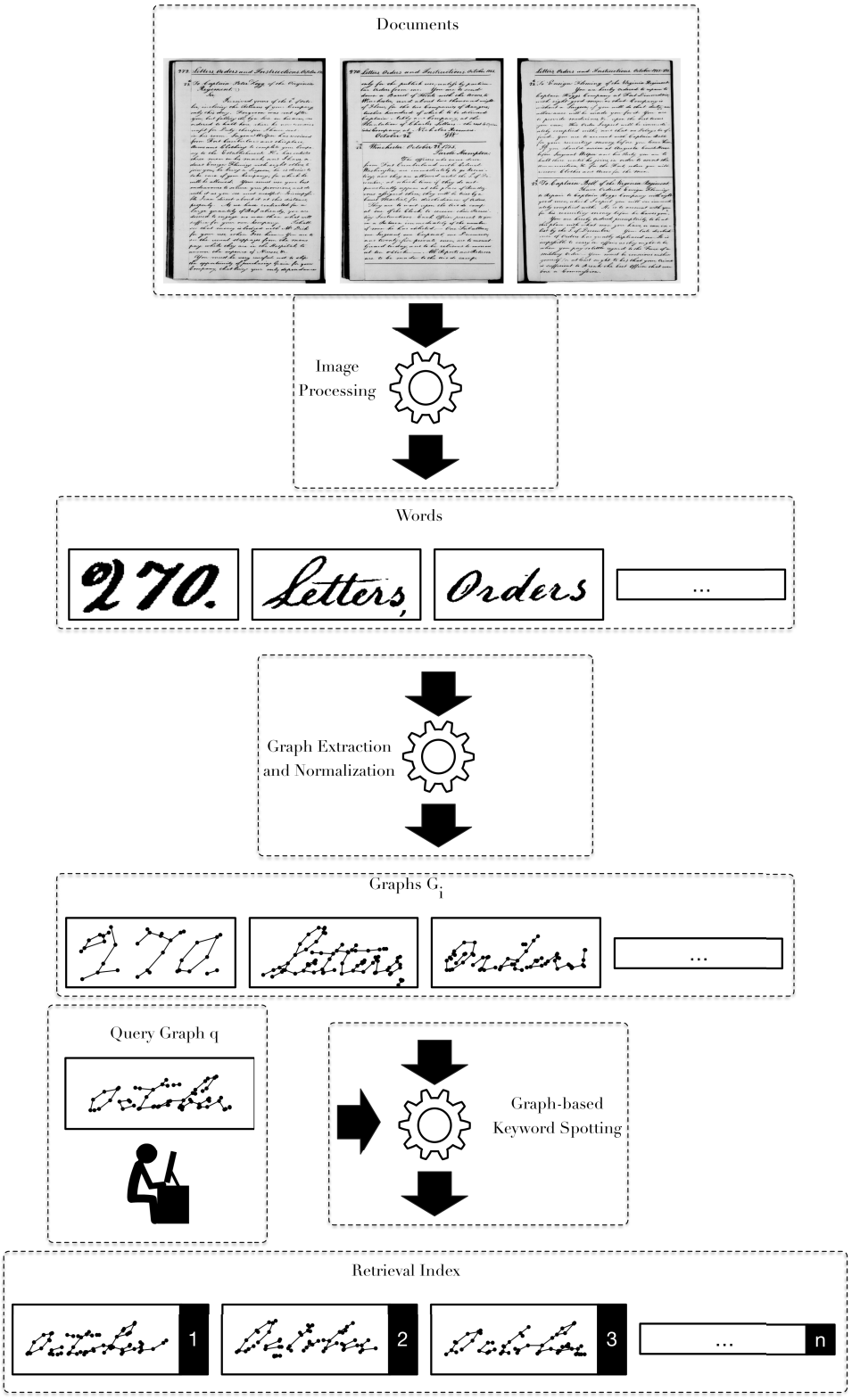


FIGURE 4.1: The graph-based KWS process.

the primary domains of interests by nodes. As a structural representation, graphs retain the binary relations between substructures with edges.

In the first step, Section 4.1.1 discusses the image processing before graph extraction. The scanned handwritten document images are binarized then segmented into word images. Section 4.1.2 takes the graph extraction into account. Concerning a single word image, four distinct graph representations are extracted. Each representation can individually be used in the KWS framework. Next, Section 4.1.3 addresses the requirement of fortifying the representation to avoid abnormalities. Hence, the postprocessing step normalizes the graphs by a z-score to minimize the abnormalities such as intraclass variations.

The graph-based keyword spotting method, Section 4.2, demonstrates the computation of graph dissimilarities to perform the task. Using graph edit distance algorithms, the KWS approach computes the distance between a specific query graph q and all document graphs $g \in G$. Finally, it builds a retrieval index from the created distances.

4.1.1 Image Preprocessing

The first step toward processing the documents is binarizing the images. When binarizing the images, it is undoubtedly necessary to detect and remove and backgrounds. Otherwise, the binarized images would have a high degree of degradation and noises.

First, the images are transformed into grayscale, if they are colored, by considering the luminance. Next, the band-pass *Difference of Gaussians (DoG)* filter is applied to the grayscale image. The DoG filter removes noises as well as enhancing the edges (Fischer, Indermühle, Bunke, Viehhauser, & Stolz, 2010). The difference of Gaussians indicates obtaining two filtered images by applying Gaussian kernels. The kernels have different standard deviation σ_1 and σ_2 and blur the images accordingly. Then one image is subtracted from the other one to obtain the DoG image. However, we have to choose

the kernels with regard to the quality of the documents. Thus the parameters σ_1 and σ_2 must be adjusted for a batch of new images to make the filter useful for extracting the foreground. Next, a global threshold T binarizes the locally enhanced document images. Figure 4.2 illustrates filtering and detecting the foreground on an example document image.

It is often required to segment the documents concerning the building blocks, such as words or characters. In a template-based method approach, we take a whole word as a recognition piece. In an unconstrained manuscript, however, the segmentation is known as an open problem. For instance, in a cursive Latin manuscript, we can acknowledge lack of precise geometrical definition for connection between characters or inclination in writing. In a top-down approach, we perform the segmentation on document images to extract first the lines and subsequently the words.

Figure 4.3 shows the horizontal projection profile on a sample document. The segmentation in Fischer, Indermühle, et al. (2010) is capable of distinguishing the boundaries by employing the projection profiles. The approach accomplishes the segmentation at a satisfactory level if the document is quasi-straight. The potential errors, without the help of the manual correction, can contribute to declining the performance. If necessary, the automatic segmentation result is manually adjusted to preclude the involvement of errors. Accordingly, the proposed keyword spotting system operates on distinct isolated words obtained from documents. Therefore, the reported performance should be taken as an upper bound on the end-to-end keyword spotting system.

We capture the structure of a word image in the graph representations. The thickness of strokes in handwriting can vary, yet the characters and lexicons shapes can remain intact regarding the topology. Therefore, a thin version of the image, namely *skeleton*, has a comparable topology with its original counterpart since it retains the strokes and connections. The *thinning* or *skeletonization* refers to the algorithmic approach of

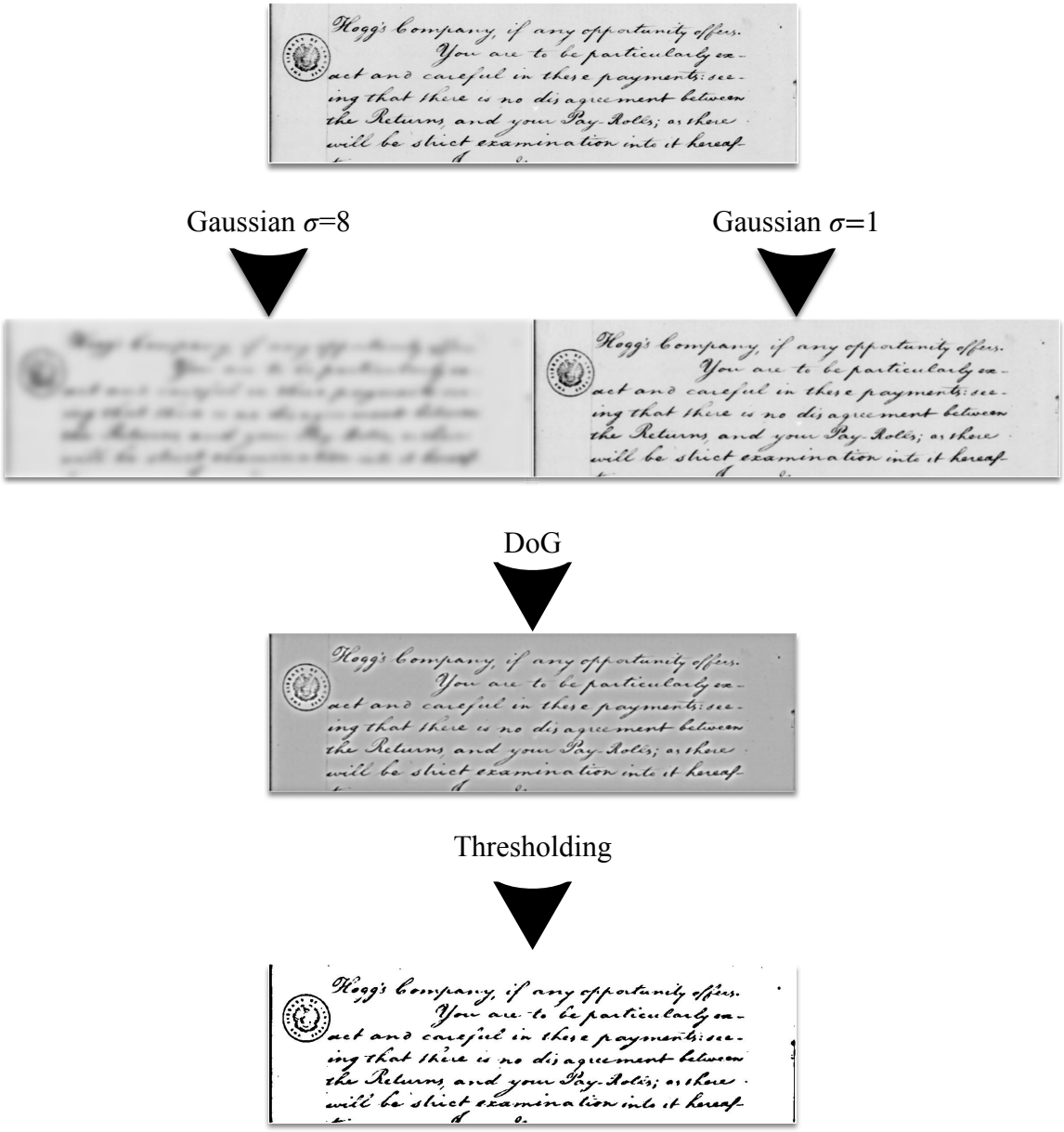


FIGURE 4.2: Preprocessing an image to obtain the foreground by applying DoG filter ($\sigma_1 = 8$ and $\sigma_2 = 1$).

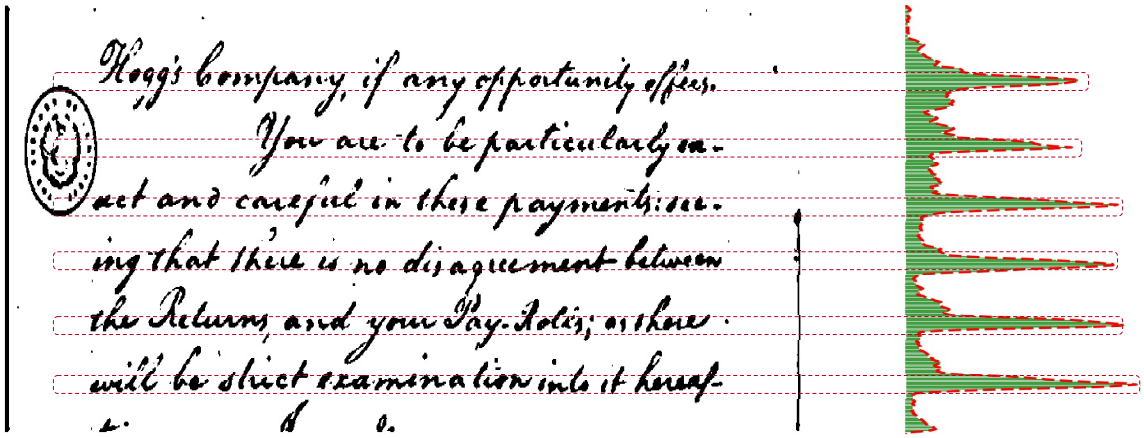


FIGURE 4.3: The horizontal projection profile of foreground pixels indicates the possibility of six text lines.

removing the boundary pixels until the thickness of the patterns is one pixel wide. Using a 3×3 thinning operator (Guo & Hall, 1989), we obtain the skeleton of the word images that is denoted by S in the forthcoming sections.

Any other thinning algorithm can substitute Guo and Hall (1989) as long as it satisfies the following conditions. The algorithm must produce a *thin* image by removing the boundary of a pattern until it is one pixel wide. However, it must not go further to eliminate the entire pixels on the images. Hence it must be *stable* and retain the skeleton. The algorithm must maintain the *connectivity* of original patterns. Finally, the *position* of skeleton must approximately be at the center of the image.

The summary of image processing in Figure 4.4 demonstrates the overview of the preparing the word images from scanned documents. The process also includes a skew correction (Hull, 1998) that is the correction of the inclination of the document. We indicate the binarized word images with B in the following sections.

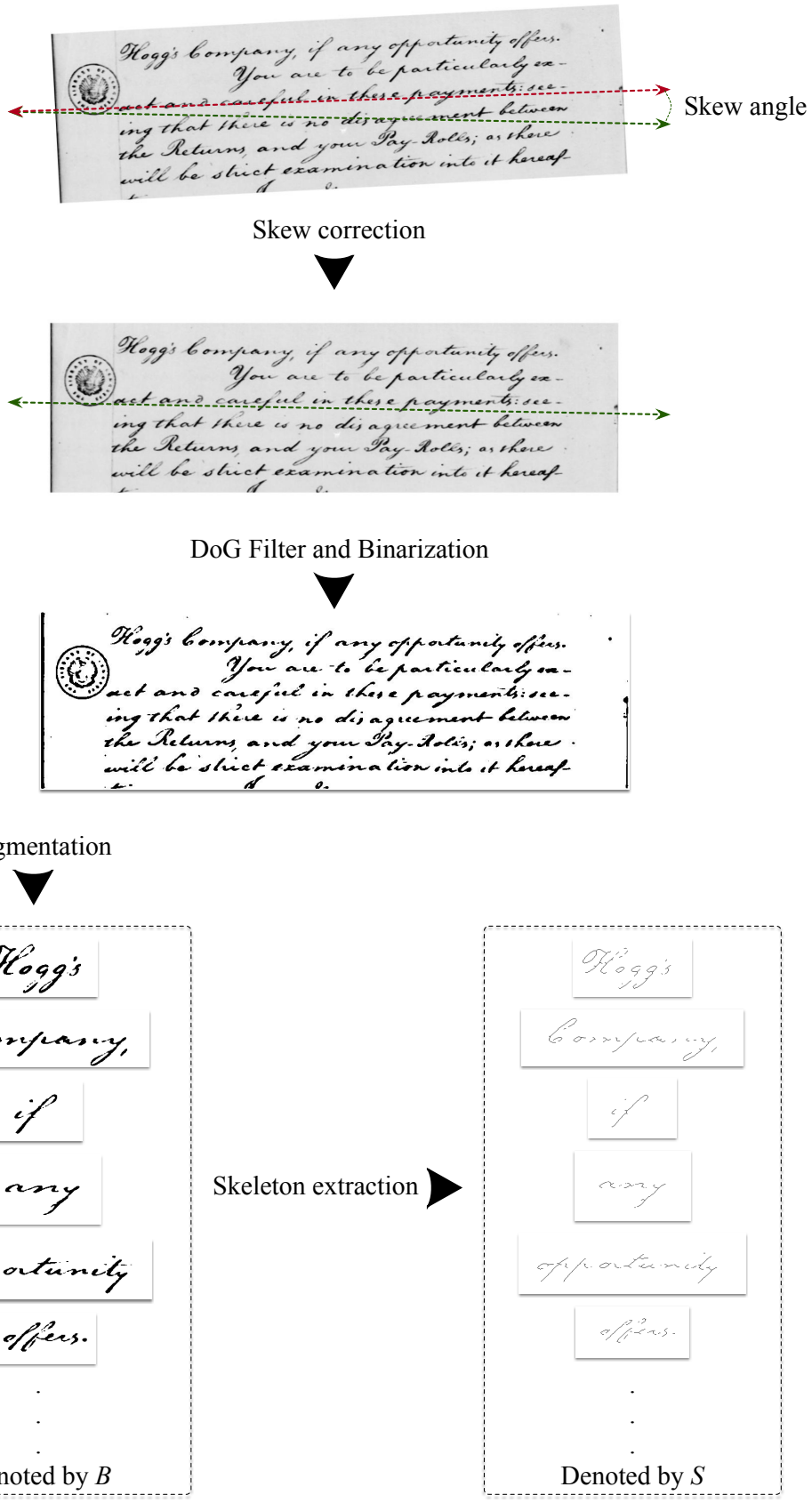


FIGURE 4.4: The image processing generates the binarized word images B and skeleton images S.

4.1.2 Graph Extraction

As a general rule in pattern recognition, related patterns must have similar representation. In the same way, different patterns should exhibit a notable dissimilarity. The rule holds true in the graph representation of handwriting. Hence, the similarity and dissimilarity determine the strength of the graph-based KWS system. The graphs of the same lexicons should not vary too much. Accordingly, the graphs representing different words must have a sizable difference. We cannot make a robust KWS system unless we satisfy these rules.

The proposed HED-based KWS represents the entire word as one graph. Although other graph-based KWS approaches in P. Wang et al. (2014a), use a collection of graphs, extracted from individual connected components, to represent a word.

In the following, we introduce and describe four different graph representations of handwriting. For further details, we refer to Stauffer, Fischer, and Riesen (2016b). All graph extraction methods, regarding definition 1, result in nodes that are labeled with two-dimensional numerical labels $L_V = \mathbb{R}^2$. The edges, however, remain unlabeled, i.e.and $L_E = \{\}$.

Keypoint graphs

The first graph extraction algorithm, *Keypoint*, makes use of characteristic points (so-called main-points) in skeletonized word images S. The keypoint graphs first used in Fischer, Riesen, and Bunke (2010) as a feature in HMM-based KWS approach. We have modified the method to take the edges into account as well.

The keypoints formally involve three categories of points: the end-points, junction-points, and upper-left-point in case of circular shapes. Figure 4.5 illustrates the main-points by

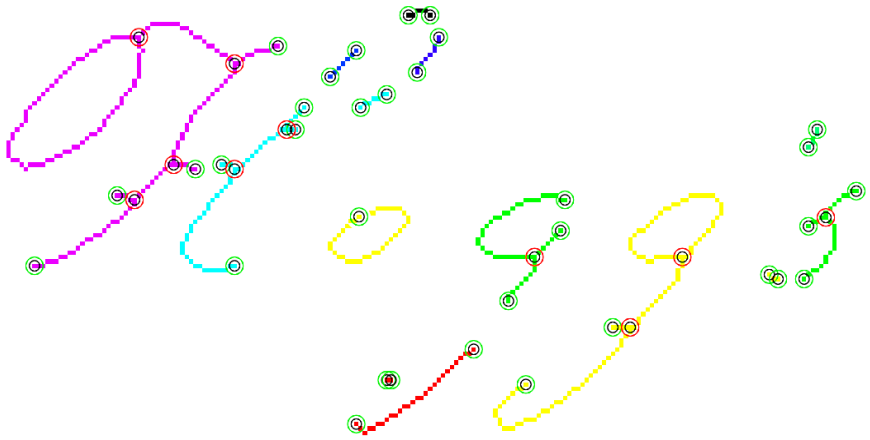


FIGURE 4.5: The keypoints for each connected component are marked with a circle.

the circles and connected components in different colors on the skeleton. The keypoints serve as the graph nodes; however, their distribution might be imbalanced.

For each *connected components (CC)*, the keypoints are first extracted from S. If multiple main-points exist nearby, a local search refines and selects one point in a region. Next, the node sets completed by connection-points. Between pairs of keypoints (on the skeleton) further intermediate points are transformed into nodes. By removing the junction points and keypoints on the upper left location of circular shapes from S, we obtain *connected subcomponents (CC_{sub})*. On each $\mathrm{CC}_{\mathrm{sub}}$, since there is no junction, we select the start and end points. Then, on an equidistant interval D, the connection-points are identified as nodes. For a chain of m points $(x_1,y_1),...,(x_m,y_m)$ on the $\mathrm{CC}_{\mathrm{sub}}$, we compute the Euclidean distance $d(i,j)$ between (x_i,y_i) and (x_j,y_j) by Eq. 4.1.

$$d(i,j) = \sum_{k=i}^{j-1} ||(x_{k+1},y_{k+1}) - (x_k,y_k)|| \tag{4.1}$$

The graph nodes are labeled with the corresponding (x,y)-coordinates on the image

where the node has been selected. Finally, undirected edges are inserted into the graph for each pair of nodes directly connected by a stroke. Algorithm 6 demonstrates the keypoint graph extraction.

Algorithm 6 Graph Extraction: Keypoints

Require: Skeleton image S, Distance threshold D.
Ensure: Keypoint graph $g = (V, E)$.

for all connected components $CC \in S$ **do**
  $V \leftarrow V \cup \{(x,y) \in CC | (x,y)$ are keypoints$\}$
  Remove junction points from CC
  for all connected subcomponents $CC_{sub} \in CC$ **do**
    $V \leftarrow V \cup \{(x,y) \in CC_{sub} | (x,y)$ on equidistant intervals $D\}$
  end for
end for
for all pairs of nodes $(u,v) \in V \times V$ **do**
  $E \leftarrow E \cup (u,v)$ if the corresponding points are connected in S
end for
return $g = (V, E)$

Grid

The second graph extraction algorithm *Grid* is based on a grid-wise segmentation. The grid-wise segmentation splits the binarized word images B into equally sized segments. The statistical features extraction techniques for keyword spotting, such as *local gradient histogram (LGH)* by Rodriguez and Perronnin (2008), and *histogram of oriented gradients (HOG)* by Almazán, Gordo, Fornés, and Valveny (2014), perform a gird-wise segmentation of word images. Grid-wise graphs extraction from the word image is illustrated in Figure 4.6.

The images $B_{w \times h}$ are segmented on the grounds of having a particular number of segments $C \times R$. The dimension of an image modifies the size of segments in the grid. The size of segments in the grid is therefore distinct based on the image dimension $w \times h$. We calculate the grid size of an image $B_{w \times h}$ with width w and height h by Eq. 4.2.

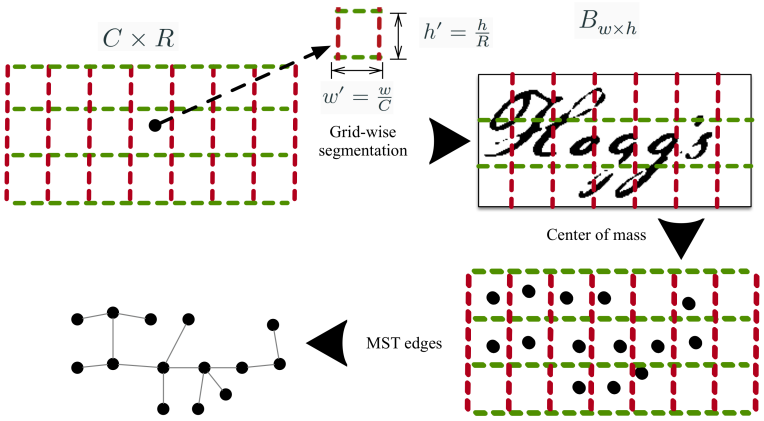


FIGURE 4.6: Grid-wise extraction of graphs.

$$w' = \frac{w}{C} \text{ and } h' = \frac{h}{R} \tag{4.2}$$

Algorithm 7 describes the graph extraction by means of grid-wise segmentation. Each segment corresponds to a potential node in the graph. A node will be assigned to a segment if it contains any foreground pixels. The (x,y)-coordinates of segment's center of mass (x_m, y_m) represent the label of the node. For an image segment with n foreground pixels, we calculate the center of mass by Eq. 4.3.

$$x_m = \frac{1}{n}\sum_{w=1}^{n} x_w \text{ and } y_m = \frac{1}{n}\sum_{w=1}^{n} y_w \tag{4.3}$$

Nonetheless, no node will be assigned to the empty segments that do not contain any foreground pixels.

For any pair of nodes u, v an undirected edge is added to E if two nodes are adjacent in the grid. Finally, the graph is transformed into *minimal spanning tree (MST)*. By trimming the graph edges, the Kruskal algorithm (Kruskal, 1956) finds the MST.

Algorithm 7 Graph Extraction: Segmentation Grid

Require: Binary image B, Grid width w, Grid height h
Ensure: graph $g = (V, E)$.

```
C ← Width of B / w                                   ▷ number of columns
R ← Height of B / h                                  ▷ number of rows
for i ← 1 : C do
    for j ← 1 : R do
        V ← V ∪ {(x_m, y_m) | (x_m, y_m) is the center of mass of segment s_ij}
    end for
end for
for all pairs of nodes (u, v) ∈ V × V do
    E ← E ∪ (u, v) if the associated segments are connected by MST
end for
return g = (V, E)
```

Projection

The next graph extraction algorithm `Projection` is computed on the horizontal and vertical projection profiles of binary images B. Projection graph extraction is illustrated in Figure 4.7.

Algorithm 8 performs a vertical and subsequent horizontal segmentation based on projection profiles. For an image $B_{w \times h}$, with width of w and height of h, the vertical projection profile $P_v = \{p_1, \ldots, p_w\}$ calculates the number of foreground pixels for every column in B. Hence the vertical projection profile comprises a sequence of w values. The image is then vertically segmented on the white spaces, locations in which there are no foreground pixels in the vicinity. Formally in a white space region, $p_i = \cdots = p_{i+k} = 0$, the image is segmented on the midpoint $p = \lfloor (p_i + p_{i+k})/2 \rfloor$ of the white space . An additional step refines the resulting segmentation concerning a distance-based threshold D_v. When the width of segment $s \in B_{segmentes}$ is greater than the threshold, it is further segmented in equidistant intervals D_v.

In the horizontal direction, the same rule applied to every vertical segment $s \in B_{segmentes}$. The horizontal projection profile $P_h = \{p_1, \ldots, p_h\}$ calculates the number of foreground

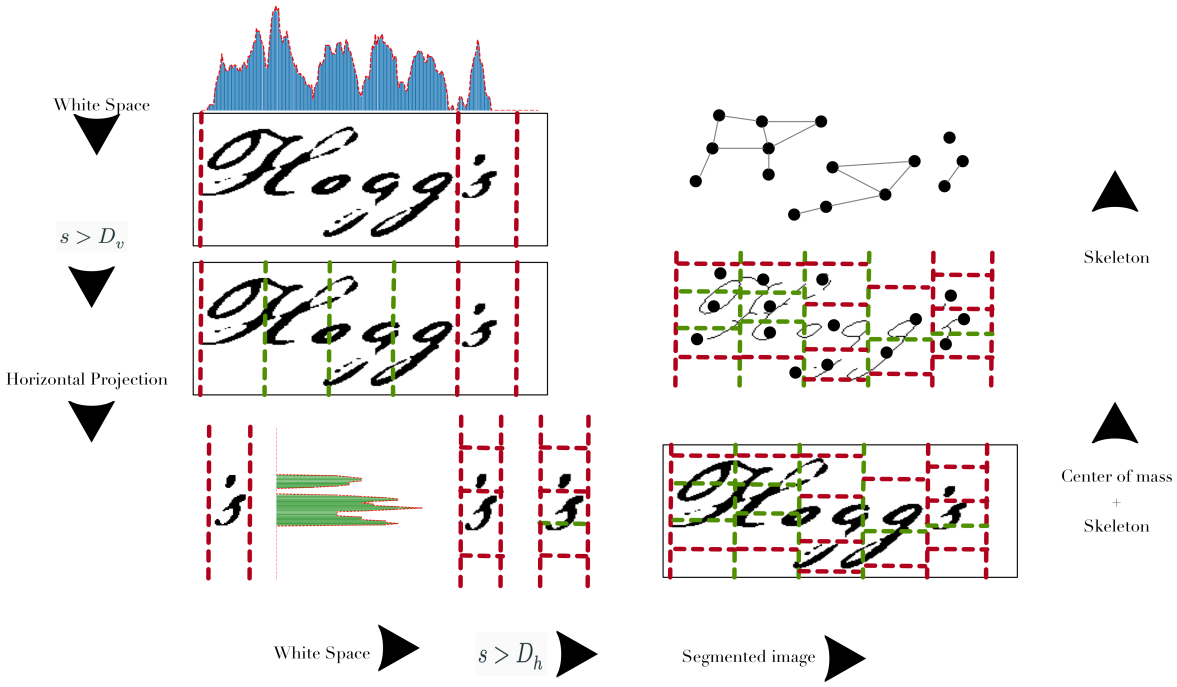


FIGURE 4.7: Projection extraction of graphs.

pixels on each row of s. In the same manner, horizontal segmentation take place on the white spaces. After splitting the segments s horizontally on the white spaces, the subsequent segments are split further on equidistant intervals D_h if the segment height is still higher than D_h. A node is inserted into the graph for each segment and labeled by the (x, y)-coordinates of the corresponding center of mass. Undirected edges are inserted into the graph for each pair of nodes if a stroke directly connects them in the skeleton S of the word image.

Split

The fourth graph extraction algorithm *split* iteratively segments the binarized word images B. Algorithm 9 computes the horizontal and vertical projection profiles similar to algorithm 8 yet on individual segments. Figure 4.8 depicts the split graph extraction.

Algorithm 8 Graph Extraction: Projection Profiles

Require: Binary image B, Skeleton image S, Vertical and Horizontal thresholds (D_v, D_h).

Ensure: graph $g = (V, E)$.

Compute vertical projection P_v of B
Split B vertically at middle of white spaces of P_v into $B_{segments}$
for all segment $s \in B_{segments}$, if Width$(s) > D_v$ **do**
    Split s vertically in equidistant intervals D_v into $B_{segments}$
end for
for all segment $s \in B_{segments}$ **do**
    Compute horizontal projection profile P_h of s
    Split s horizontally at middle of the white spaces of P_h into $B_{segments}$
    for all segment $s \in B_{segments}$, if Height$(s) > D_h$ **do**
        Split s horizontally in equidistant interval D_h into $B_{segments}$
    end for
end for
for all segment $s \in B_{segments}$ **do**
    $V \leftarrow V \cup \{(x_m, y_m) | (x_m, y_m)$ is the center of mass of segment $s\}$
end for
for all pairs of nodes $(u, v) \in V \times V$ **do**
    $E \leftarrow E \cup (u, v)$ if the associated segments are connected in S
end for
return $g = (V, E)$

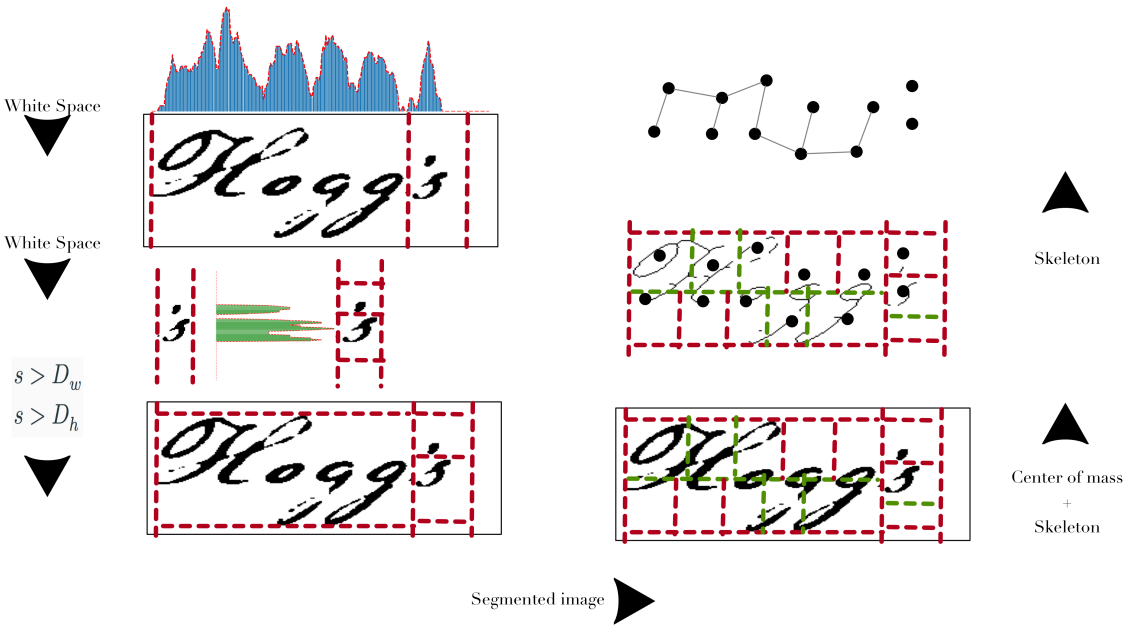


FIGURE 4.8: Split extraction of graphs.

The algorithm 9 begins with one segment $s = B$ that considers the whole image as an initial segment. Then, each $s \in B_{segments}$, having a width w greater than the width threshold D_w, splits vertically. The split point of segment s is at the white space of projection profile if the white space exists. The vertical midpoint of s determines the split point accordingly if no white space can be located.

In the same way, after a pass of vertical segmentation, the height of segments are compared with D_h. Each segment $s \in B_{segmentes}$ that has a height $h > D_h$ is divided on horizontal white space or the midpoint. That is, segments are iteratively split into smaller subsegments until the width and height of all segments are below certain thresholds.

A node is inserted into the graph and labeled by the (x,y)-coordinates of the point on the stroke closest to the center of mass of each segment. For the insertion of the edges, the same procedure as for `Projection` is applied.

4.1.3 Graph Normalization

A sequence of operations processes the scanned documents to produce the handwriting graphs. A small deviation on each stage can expand into a significant variation of the final graph. To moderate the variations of the intraclass writing, we normalize the graphs Figure 4.9.

The resulting set of graphs is normalized concerning the (x,y)-coordinates of their node labels $\mu(v)$. Formally, we use a z-score to derive normalized coordinates $(\hat{x},\hat{y})$ by

$$\hat{x} = \frac{x - \mu_x}{\sigma_x} \text{ and } \hat{y} = \frac{y - \mu_y}{\sigma_y} \quad ,$$

where (μ_x, μ_y) and (σ_x, σ_y) are the mean and standard deviation of all (x,y)-coordinates in the graph under consideration.

Algorithm 9 Graph Extraction: Split

Require: Binary image B, Skeleton image S, Width threshold D_w, and Height thresholds D_h.
Ensure: graph $g = (V, E)$.

$B_{segments} \leftarrow B$
while Width(s) $> D_w$ or Height(s) $> D_h$ for any segment $s \in B_{segments}$ **do**
  for all segment $s \in B_{segments}$, if Width(s) $> D_w$ **do**
    if s contains white space in vertical projection profile P_v **then**
      Split s vertically at the middle of white space of P_v into $B_{segments}$
    else
      Split s vertically at the center of s into s_1, s_2
    end if
  end for
  for all segment $s \in B_{segments}$, if Height(s) $> D_h$ **do**
    if s contains white space in horizontal projection profile P_h **then**
      Split s horizontally at the middle of white space of P_h into $B_{segments}$
    else
      Split s horizontally at the center of s into s_1, s_2
    end if
  end for
end while
for all segment $s \in B_{segments}$ **do**
  $V \leftarrow V \cup \{(x_m, y_m) | (x_m, y_m)$ is the center of mass of segment $s\}$
end for
for all pairs of nodes $(u, v) \in V \times V$ **do**
  $E \leftarrow E \cup (u, v)$ if the associated segments are connected in S
end for
return $g = (V, E)$

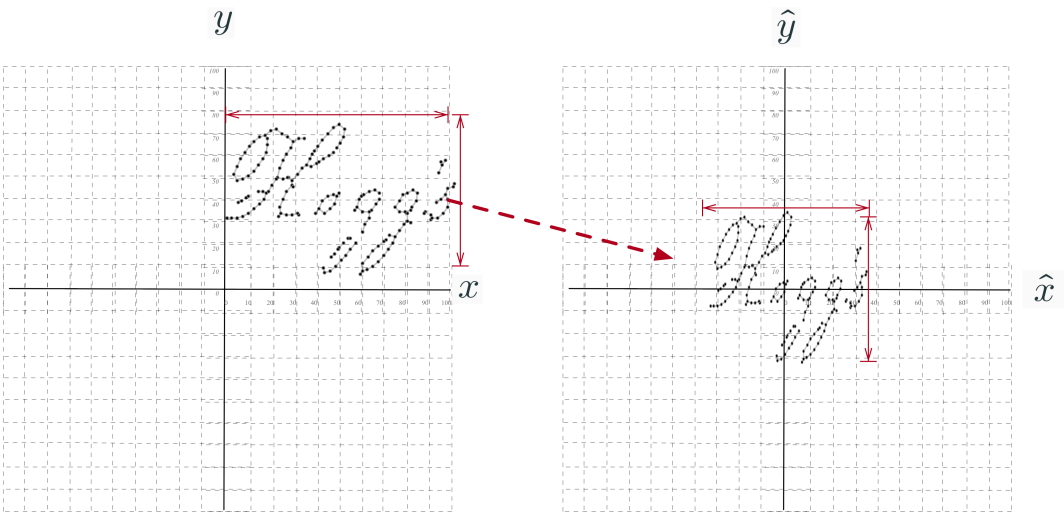


FIGURE 4.9: Normalizing the extracted graphs.

4.2 Bipartite Graph Matching for Keyword Spotting

So far we discussed the image processing and graph extractions of the graph-based KWS. The final step performs graph matching to calculate the retrieval index of the relevant words.

In graph-based KWS, a query graph q (representing a particular keyword) is pairwise matched against all document graphs $G = \{g_1, \ldots, g_N\}$. Generally, graphs can either be matched utilizing exact or inexact approaches (Conte et al., 2004; Foggia et al., 2014). In the case of graph-based KWS, graphs are used to represent the inherent characteristic of handwriting, and thus, affected by (subtle) variations in both their structure and labels. Accordingly, as explained in chapter 3, inexact graph matching can be applied only.

Several approaches have been proposed for inexact graph matching (Conte et al., 2004; Foggia et al., 2014). The graph edit distance (GED) is regarded as one of the most flexible and robust paradigms (Bunke & Allermann, 1983; Riesen, 2015).

As discussed in chapter 3, the exact computation of GED is exponential concerning the number of nodes of the involved graphs. Formally, GED is an example of a *Quadratic Assignment Problem (QAP)* (Koopmans & Beckmann, 1957). The QAP problems, and consequently GED, belong to the class of $\mathcal{NP}$-complete problems [1].

The graph-based KWS in Stauffer et al. (2016a) has used the BP algorithm (Riesen & Bunke, 2009a) to approximate the GED. The BP algorithm finds the optimal matching between nodes and their connected edges structures, yet the solution is suboptimal with respect to the global graph structure. The BP algorithm reduces the QAP problem to LSAP by disregarding the global graph structure. Hence, it enhances the complexity to the cubic time.

[1] That is, an exact and efficient algorithm for the graph edit distance problem cannot be developed unless $\mathcal{P} = \mathcal{NP}$.

The BP distance d_{BP} is calculated between query q and the word graphs $G = \{g_1, \ldots, g_N\}$. The graphs from various classes of words are different is size. Consequently, d_{BP} must be interpreted regarding the graphs sizes. The accommodation is done with the normalization term in the denominator of Eq. 4.4. The normalization term is associated with the worst case in graph matching where every node and edge is deleted or inserted into the edit path. Finally, the revival score $r(q,g)$ in Eq. 4.4 accounts for the priority of retrieved word in descending order.

$$r(q,g) = -\frac{d_{\mathrm{BP}}(q,g_i)}{(|V_q| + |V_{g_i}|)\,\tau_v + (|E_q| + |E_{g_i}|)\,\tau_e} \tag{4.4}$$

Chapter 5

The Proposed KWS Systems

This chapter is based on the following published articles:

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T., & Fischer, A. (2017). Keyword Spotting in Historical Documents Based on Handwriting Graphs and Hausdorff Edit Distance. In *International graphonomics society conference* (pp. 105–108).

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T. D., & Fischer, A. (2018). Graph-based keyword spotting in historical manuscripts using hausdorff edit distance. *Pattern Recognition Letters*.

In this chapter, the proposed KWS systems are presented and empirically evaluated. The HED-based KWS system in section 5.1 employs, HED, a fast graph matching algorithm in order to improve the performance of graph-based KWS. The combined HED-DTW-based KWS system is presented in section 5.2 as a multi-classifier approach which benefits from both structural and statistical representation of handwriting. The datasets which we use to evaluate systems are described in section 5.3. In order to evaluate the KWS approaches, we must consider the design of methods and types of data which are being processed. Section 5.4 describes and discussed the choice of evaluation metrics. Section 5.5 is dedicated to the numerical demonstration of the proposed method. The selection of parameters as well as sample outputs are explained in detail. We compare

the performance of the proposed method with three template-based reference methods, namely BP, BP2, and DTW. We also put our method into context with learning-based approaches to keyword spotting. All of which are detailed in section 5.6. In section 5.7, we demonstrate the efficiency of the proposed graph-based keyword spotting. We evaluate the proposed HED-based method on four benchmark datasets for keyword spotting in historical manuscripts. The results have been presented and published at Ameri et al. (2017, 2018). Finally, the conclusion is presented in section 5.8.

5.1 Keyword Spotting based on HED

In this section, we describe a graph-based KWS, as a template-matching method, which does not require any learning. The graph-based KWS, as well as any template-based approach, in the worst condition, expect at least a single template image of the keyword. Although with a more substantial number of keyword templates, the system has more option to incorporate the variation of intraclass words. This is particularly useful for historical manuscripts, which typically demand human experts for obtaining labeled training data in a time-consuming and costly process. The graph-based keyword spotting in Figure 4.1 aims at finding similar words to a query keyword in the form of a retrieval index.

The GED graph matching algorithm provides flexibility to match arbitrary graphs. In particular, GED measures the amount of distortion needed to transform graph g_1 into graph g_2 using a sequence of edit operations like *insertions*, *deletions*, and *substitutions* of both nodes and edges. The sequence is referred to as *edit path*, $\lambda(g_1, g_2)$, between g_1 and g_2.

To find the most suitable edit path, one commonly introduces a particular cost function $c(e)$ for every edit operation e. This cost function should correspond to the strength

of a specific graph modification. Therefore, the graph edit distance $d_{\text{GED}}(g_1, g_2)$, or d_{GED} for short, between g_1 and g_2 comprises the sum of edit path costs (see Eq. 3.3).

For the representation of domain knowledge, one commonly makes a change to Eq.3.9-3.10 to obtain an appropriate cost model. In the current case, constant costs for both node and edge deletions/insertions are used, i.e. $\tau_n \in \mathbb{R}^+$ and $\tau_e \in \mathbb{R}^+$, respectively. For the substitution of nodes, however, we adjust Eq.3.9 to the weighted Euclidean distance between the corresponding node labels, i.e., $\delta = (x, y)$-coordinates. Formally,

$$C(v_i \rightarrow v_j) = \alpha . \sqrt{\beta\, \sigma_x (x_i - x_j)^2 + (1-\beta)\, \sigma_y (y_i - y_j)^2} \tag{5.1}$$

where $\beta \in [0,1]$ denotes a parameter to weight the importance of the x- and y-coordinate of a node. The actual z-score values μ and σ are retained when normalizing graphs. Hence, σ_x and σ_y denote the standard deviation of all node coordinates in the current query graph before normalization. Moreover, we still use a weighting factor $\alpha \in [0,1]$ between the node and edge edit costs.

Hence, we can use fast, but suboptimal algorithms, that have been proposed in the last years (see Foggia et al. (2014)). In this thesis, we consider the recently introduced Hausdorff edit distance (HED) (Fischer et al., 2015). HED reduces the problem of graph edit distance to a set matching problems between local substructures (nodes and their adjacent edges). In section 3.6, we have demonstrated HED could be computed in quadratic time concerning the graph size. The suboptimal computed distance, yet is a lower bound of graph edit distance $d_{\text{HED}} \leq d_{\text{GED}}$.

The Hausdorff edit distance $d_{HED}(g_1, g_2)$ between two graphs g_1 and g_2, using Eq. 3.16, is obtained in Eq. 5.2 as :

$$d_{HED}(g_1, g_2) = \sum_{u \in V_1} \min_{v \in V_2 \cup \{\varepsilon\}} f(u,v) + \sum_{v \in V_2} \min_{u \in V_1 \cup \{\varepsilon\}} f(u,v) \quad . \tag{5.2}$$

Similar to the Hausdorff distance between finite subsets of a metric space, the two summation terms compute nearest neighbor distances between the node sets. According to the node function Eq. 3.17-3.19 , using constant cost τ_n and τ_e for insertion/deletion of nodes and edge respectively, we obtain the node cost function by Eq. 5.3.

$$f(u,v) = \begin{cases} \tau_n + \sum_{i=1}^{|P|} \frac{\tau_e}{2} & \text{for node deletion } (u \to \varepsilon) \\ \tau_n + \sum_{i=1}^{|Q|} \frac{\tau_e}{2} & \text{for node insertion } (\varepsilon \to v) \\ \frac{c(u \to v) + \frac{d_{HED}(P,Q)}{2}}{2} & \text{for node substitution } (u \to v) \end{cases} \tag{5.3}$$

The cost function regards P and Q as the set of edges adjacent to u and v, respectively. Note that only half of the implied edge cost is added to the node cost and only half of the substitution cost is considered in general as explained in the original cost function, to ensure the lower bound property.

The edge cost Eq. 3.23, which is implied by node substitution, is estimated based on the edge sets P and Q. With a similar Hausdorff matching using edge cost function g, we obtain $d_{HED}(P,Q)$ in Eq. 5.4.

$$d_{HED}(P,Q) = \sum_{p \in P} \min_{q \in Q \cup \{\varepsilon\}} g(p,q) + \sum_{q \in Q} \min_{p \in P \cup \{\varepsilon\}} g(p,q) \tag{5.4}$$

The edge functions Eq. 3.20-3.22 , are accordingly adjusted with constant cost τ_e to Eq.5.5.

$$g(p,q) = \begin{cases} \tau_e & \text{for edge deletion } (p \to \varepsilon) \\ \tau_e & \text{for edge insertion } (\varepsilon \to q) \\ \frac{c(p \to q)}{2} & \text{for edge substitution } (p \to q) \end{cases} \tag{5.5}$$

The underestimation of $d_{HED} \leq d_{GED}$ is limited by a minimum edit costs given by Eq. 3.24 and Eq. 3.25. According to the constant cost functions they can be calculated by $||V_1| - |V_2|| \cdot \tau_n$ for $d_{HED}(g_1, g_2)$ and $||P| - |Q|| \cdot \tau_e$ for $d_{HED}(P, Q)$.

The retrieved documents are selected based on the KWS score. For building the KWS score, the graph-based KWS computes the distance between query graph q and all document graphs $G = \{g_1, \ldots, g_N\}$ using approximate graph edit distances d_{HED}. The distance is normalized by the maximum cost edit path between q and g_i. Hence, the normalization term is calculated by deleting all nodes and edges of q and inserting all nodes and edges in g_i that corresponds to the edit path with the maximum cost. Eventually, using Eq. 5.6, we calculate the rank scores $r(q, g)$ to determine the index of retrieved documents.

$$r(q, g) = -\frac{d_{\text{HED}}(q, g_i)}{(|V_q| + |V_{g_i}|)\,\tau_v + (|E_q| + |E_{g_i}|)\,\tau_e} \tag{5.6}$$

$r(q, g)$ scores are used to determine the retrieved words with highest values. The range of $r(q, g)$ values is $r(q, g) \in [-1, 0]$ where the highest value of $r(q, g) = 0$ indicates that q and g are identical. The lowest value $r = -1$ corresponds to considerable dissimilarities in the graph where there is no substitution. Thus, the cost of d_{HED} consists deletion and insertion of graph nodes and edges that is the same as the denominator.

A query can consist of a set of graphs $Q = \{q_1, \ldots, q_t\}$ where all $q \in Q$ represent the same keyword. The minimal graph edit distance achieved on all t query graphs accordingly determines the KWS score.

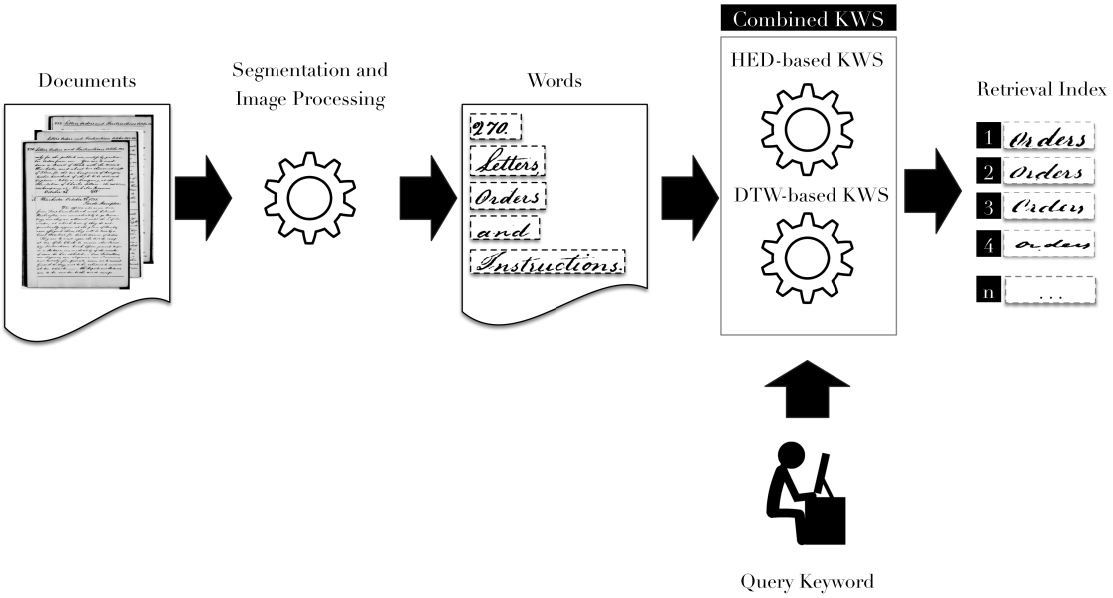


FIGURE 5.1: Combined keyword spotting workflow

5.2 Combined Graph-Based and DTW-Based Keyword Spotting

In this section, we propose a combined HED-DTW keyword spotting approach. Figure 5.1 demonstrates the architecture of the system. The matching score is computed by combining the HED (section 5.1) and DTW (section 2.1). The two methods are different, one is matching two-dimensional graphs, and the other is matching one-dimensional sequences. Feature representation of DTW-based system includes nine-Geometric features Figure 5.2. The difference in solving the problem indicates they can complement each other. They have a high potential to support each other in the *multiple classifier systems (MCS)* considering the complementary properties. In such an MCS setting, ideally, one method can correct errors of the other method (Kuncheva, 2004).

Both approaches rank the similar words to the query by providing a spotting score. Hence, we combine the scores with a weighted sum of d_{HED} and d_{DTW} to obtain the MCS score. The calculation of MCS scores for the combined system is demonstrated in Figure 5.3.

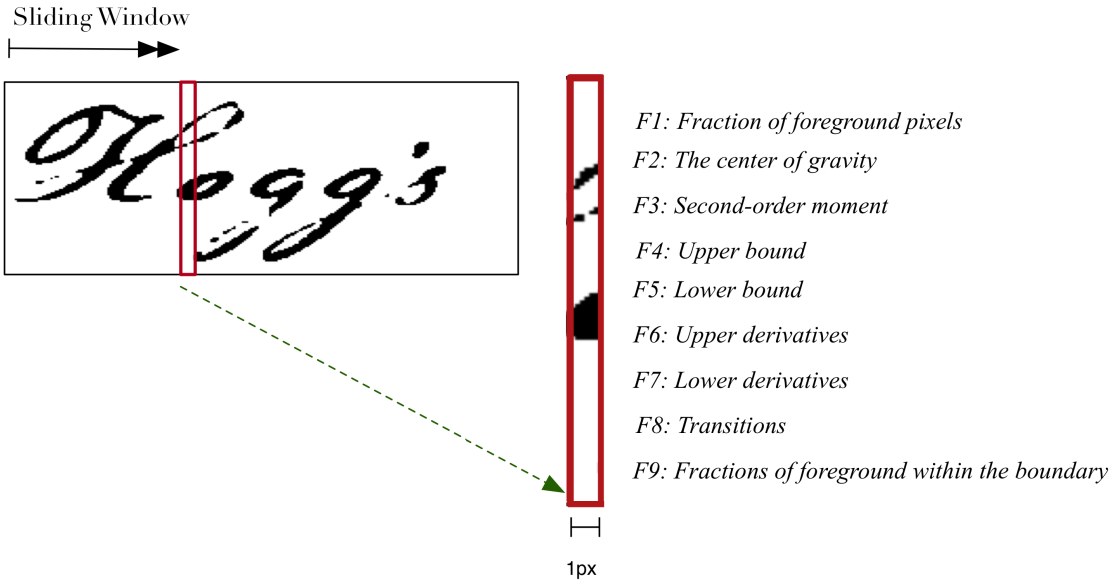


FIGURE 5.2: Geometric feature extraction.

First, we normalize the HED and DTW scores to zero mean and unit standard deviation. The z-score is calculated for each algorithm separately in the validation stage. The normalized HED and DTW scores, $r_{HED}(q,w)$ and $r_{DTW}(q,w)$ respectively, are combined with a weighted sum given in Eq. 5.7

$$r_{combined}(q,w) = r_{HED}(q,w) + \omega \cdot r_{DTW}(q,w) \tag{5.7}$$

where the weight parameter ω expresses the contribution of each score to the rank $r_{combined}(q,w)$. The weighted sum calculation typically assigns independent weights to the components. There is a possibility of using a single parameter as a pair $(\omega, 1-\omega)$ for binary relations. However, we had the idea of designing a flexible combined system to use multiple recognition engines. In our case we observed by using a fixed weight for r_{HED}, the optimal parameter can be obtained by examining a single ω weight. In section 5.5 we empirically demonstrate the correlation of weights and potential range of ω.

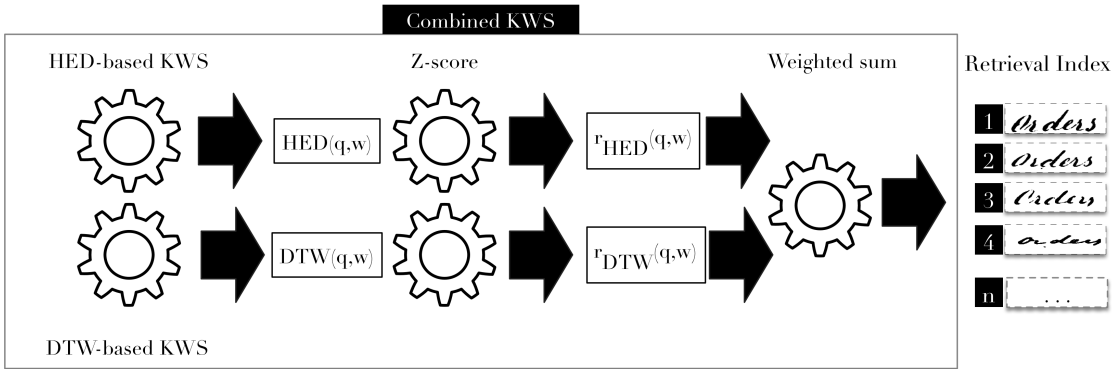


FIGURE 5.3: Score Calculation in combined system

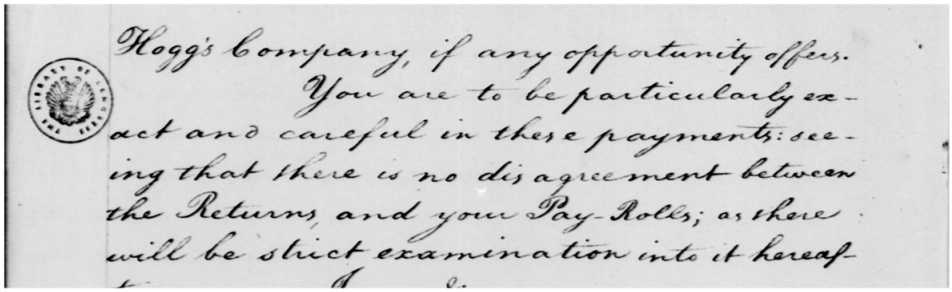


FIGURE 5.4: An example handwriting taken from George Washington dataset.

5.3 Datasets

George Washington The Library of Congress keeps the record of 65,000 manuscripts written by George Washington and his associates between 1741-1977. The *George Washington (GW)* [1] dataset consists of letters of George Washington and his associates during the American Revolutionary War in 1755. The letters are written in English with minor variations in writing and degradation Figure 5.4. It is based on twenty pages of letters which contain 4994 words in total.

Parzival The *Parzival (PAR)*[2] is based on stories of the German poet Wolfgang von Eschenbach in the 13th century. The manuscript is written in Middle High German on

[1] George Washington Papers at the Library of Congress, 1741-1799: Series 2, Letterbook 1, pp. 270-279 & 300-309, `http://memory.loc.gov/ammem/gwhtml/gwseries2.html`

[2] Parzival at IAM historical document database, `http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/parzival-database`

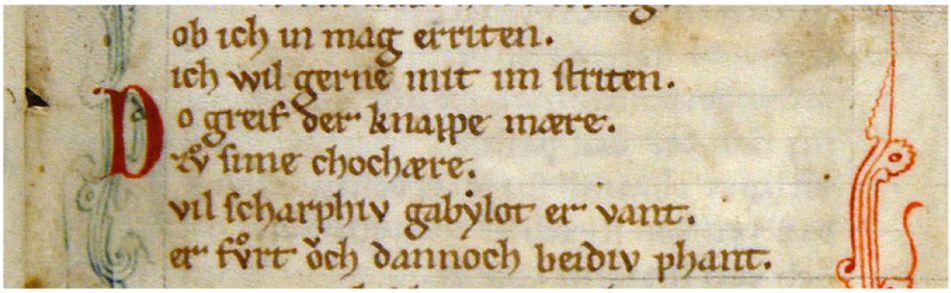


FIGURE 5.5: An example handwriting taken from Parzival dataset.

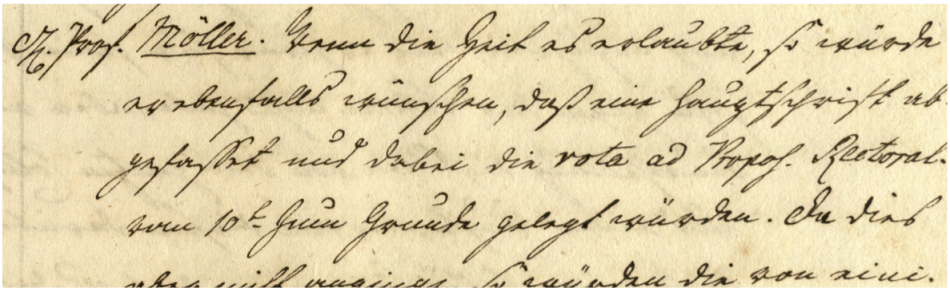


FIGURE 5.6: An example handwriting taken from Alvermann Konzilsprotokolle dataset.

parchment. From 45 pages, a total of 23 478 words have been extracted. The manuscripts have been written with low writing variations, by three writers, yet they have markable signs of degradation. The image in Figure 5.5 illustrates a page segment of Parzival dataset.

Alvermann Konzilsprotokolle The *Alvermann Konzilsprotokolle (AK)*[3] consists of minutes of formal meetings held by the central administration of the University of Greifswald from 1794 to 1797. The notes were written in German and based on 18 000 pages with minor variations and signs of degradation. Figure 5.6 shows a page segments captured from Alvermann Konzilsprotokolle dataset.

[3] Alvermann Konzilsprotokolle at ICFHR2016 benchmark database, `http://www.prhlt.upv.es/contests/icfhr2016-kws/data.html`

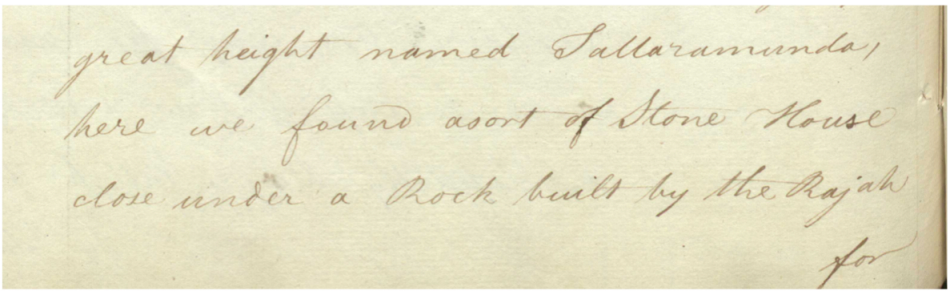


FIGURE 5.7: An example handwriting captured from Botany dataset.

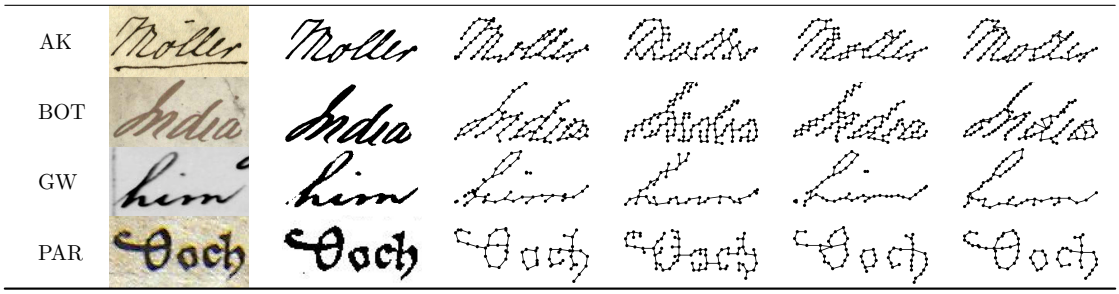


FIGURE 5.8: Exemplary graph representations of the Alvermann Konzilsprotokolle (AK), Botany (BOT), George Washington (GW), and Parzival (PAR) dataset.

Botany Finally, the *Botany (BOT)*[4] is based on botanical records made in British India in the 18th and 19th centuries. The records were written in English and based on ten pages with high writing variation and markable signs of degradation. Figure 5.7 shows a page segment of Botany dataset.

For the experimental evaluation, we consider two well-known manuscripts as well as two documents of a very recent KWS benchmark competition. On all four manuscripts, we extract graphs by means of the graph representation formalisms proposed in Section 4.1. Note that for AK and BOT, only the two most promising graph representations (`Keypoint` and `Projection`) are considered.

Figure 5.8 shows an exemplary word of each manuscript and the corresponding graph representation.

[4]Botany at ICFHR2016 benchmark database, `http://www.prhlt.upv.es/contests/icfhr2016-kws/data.html`

5.4 Evaluation Metrics

The evaluation metrics in machine learning and pattern recognition applications determine the efficiency of methods. Researchers evaluate systems on several criteria. An evaluation measure might describe the effectiveness of a paradigm yet it could be trivial on the others. Thus the choice of evaluation metrics should guide us to evaluate the system in general aspects.

Keyword spotting systems operate on a collection of documents and respond to the user queries. The response contains a group of words similar to the keyword. The *retrieved* words are ranked based on the similarities to the query keyword. The ground truth of test data specifies the binary relation of *relevant* or *nonrelevant* between any query-word pair in the collection. The confusion matrix in table 5.1 describes the four fundamental metrics, *true positive (TP)*, *false positive (FP)*, *true negative (TN)*, and *false negative (FN)* in terms of the number of retrieved/not-retrieved and relevant/nonrelevant words.

TABLE 5.1: The TP, FP, TN, FN measures based on the number of retrieved/not-retrieved and relevant/nonrelevant

	Number of relevant	Number of nonrelevant
Number of retrieved	TP	FP
Number of non-retrieved	FN	TN

Common measure such as *accuracy (ACC)* is the percent of correctly classified samples in a classification $ACC = (TP + TN)/(TP + FP + TN + FN)$. The measure might be helpful in character and text recognition tasks, yet it is less useful in keyword spotting frameworks. Since the number of samples mostly skewed toward nonrelevant words, TN contributes a considerable portion in the accuracy. Thus a system that optimized based on accuracy tends to work well in reporting non-keywords. The strategy to reject every word as non-keyword can score a high accuracy value. In contrast, a significant improvement in TP does not reflect much on the accuracy.

Considering the disadvantages of accuracy, we would focus on metrics to better reflect the positive cases. The *precision (P)* $P = TP/(TP+FP)$ is the ratio of relevant and retrieved words, TP, over retrieved words. In a typical response the precision calculates the percentage of the correct answer in the response. If we restrict the response to the highest possible scores, we get higher precision, yet the response may not cover lots of relevant keywords in the document. In other words, relevant keywords could be a small fraction of all keywords that exist in the document.

The *recall (R)*, also called *sensitivity*, $R = TP/(TP+FN)$ measures the percent of relevant words in the response. Recall on the other hand measures how much of relevant data has been retrieved. A high value of recall is proportional to having a significant number of positive values. However, the recall individually does not signify the quality of the method. Nonetheless, by returning all words in the response the recall would have the value of $R = 1$, yet it is a response carrying no information.

A keyword spotting system with high quality must take precisions and recalls into account. To combine the two measure one might suggest averaging the precision and recall values, however, considering the trivial cases the average could be 0.5 when recall is $R = 1$ and precision is $P = 0$. The *F measure* is weighted harmonic mean of precision and recall. Unlike the arithmetic mean *F measure* converges to the smaller value of precision or recall, hence provides a more robust metric.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \tag{5.8}$$

The value $\alpha \in [0,1]$ specifies the balance between precision and recall. In case of equal importance $\alpha = 1/2$, precision has equal importance as recall[5]. The method is at peak performance where the F measure on the precision-recall curve is maximized.

[5]The *F measure* is alternatively characterize by $\beta \in [0,\infty)$, as $F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P+R}$. The β parameter therefore is $\beta^2 = \frac{1-\alpha}{\alpha}$. The $\beta > 1$ considers higher priority to recall and $\beta < 1$ emphasize more on precision.

However, the F measure is calculated by precision at a specific recall. Investigating the precision change with respect to recall is most suited for the ranked output of keyword spotting systems. The *precision-recall curve* demonstrates the variation of precision by changing the sensitivity. It is ideal to investigate the entire curve rather than an individual point that maximizes F measure. The curve would include a large number of recall points, regarding the retrieved response. The information retrieval scholars would instead summarize the evaluation metrics to fewer points, National Institute of Standards and Technology (2009) suggests *11-point interpolated average precision* as a suitable measure to evaluate ranked outcomes.

Considering the 11 recall points $r \in \lambda = \{0.0, 0.1, \ldots, 1.0\}$, the corresponding precision $p(r)$ is shifted toward the maximal value in higher recall points. Thus the precision $p(r)$ at recall r, is calculated by $p(r) = \max_{r' \geq r} P(r')$. The *interpolated average precision (AP)* refers to the arithmetic mean of these 11 precision values. Furthermore, the precision-recall curve is interpolated consequently with 11-point precision. The AP value, therefore, interpolates the area under the curve for precision-recall curve.

$$AP = \frac{1}{|\lambda|} \sum_{r \in \lambda} p(r) \tag{5.9}$$

The extensive evaluation requires testing system on a diverse set of keyword queries $Q = \{q_1, \ldots, q_n\}$. The average precision evaluates the retrieved ranked words for the query $q \in Q$ as $\mathrm{AP}(q)$. The *mean average precision (MAP)* aggregates the individual average precision $\mathrm{AP}(q)$ by arithmetic averaging in Eq. 5.10.

$$\mathrm{MAP}(Q) = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{5.10}$$

The MAP value has been used in this thesis to evaluate the keyword spotting approach extensively. MAP summarizes the precision-recall curve to a single value. To evaluate

the keyword spotting performance, we consider Recall and Precision for each keyword query and compute the (MAP) over all queries using the `trec_eval`[6] software.

5.5 Parameter Optimization

This section demonstrates setting up the proposed methods with the parameters in order to achieve the optimized system. Our proposed systems are template based approaches that do not require a training stage; however, they require a set of parameters that must be initialized. The experiments in this sections are conducted on a small set of data including five query words (*Colonel, now, no, soon, October*) and 100 randomly selected words from GW dataset.

5.5.1 Connection Node Distance in Keypoint Graphs

TABLE 5.2: Evaluation metrics for connection points selection

D	MAP	F1	ACC	P	R	TP	TN	FP	FN
1	85.72	84.29	97.00	83.33	85.78	85.78	98.23	1.77	14.22
2	85.97	82.75	97.20	92.42	77.56	77.56	99.34	0.66	22.44
3	86.58	82.91	97.40	97.78	75.11	75.11	99.78	0.22	24.89
4	83.26	79.22	96.60	90.28	73.11	73.11	99.12	0.88	26.89
5	83.63	80.95	97.00	95.00	73.11	73.11	99.56	0.44	26.89
6	83.17	81.99	95.80	83.85	85.33	85.33	96.89	3.11	14.67
7	78.27	77.61	93.40	70.25	93.56	93.56	93.34	6.66	6.44
8	75.82	75.78	94.20	71.46	83.56	83.56	95.35	4.65	16.44
9	73.38	73.71	92.20	65.14	91.56	91.56	92.24	7.76	8.44
10	74.87	72.60	94.00	70.31	81.78	81.78	95.36	4.64	18.22

Keypoint graphs consist of the main points and connection points (also named auxiliary points). The connection points are inserted into the graph on monotonic intervals. The optimal interval is adjusted by investigating MAP values of experiments which are

[6] `http://trec.nist.gov/trec_eval`

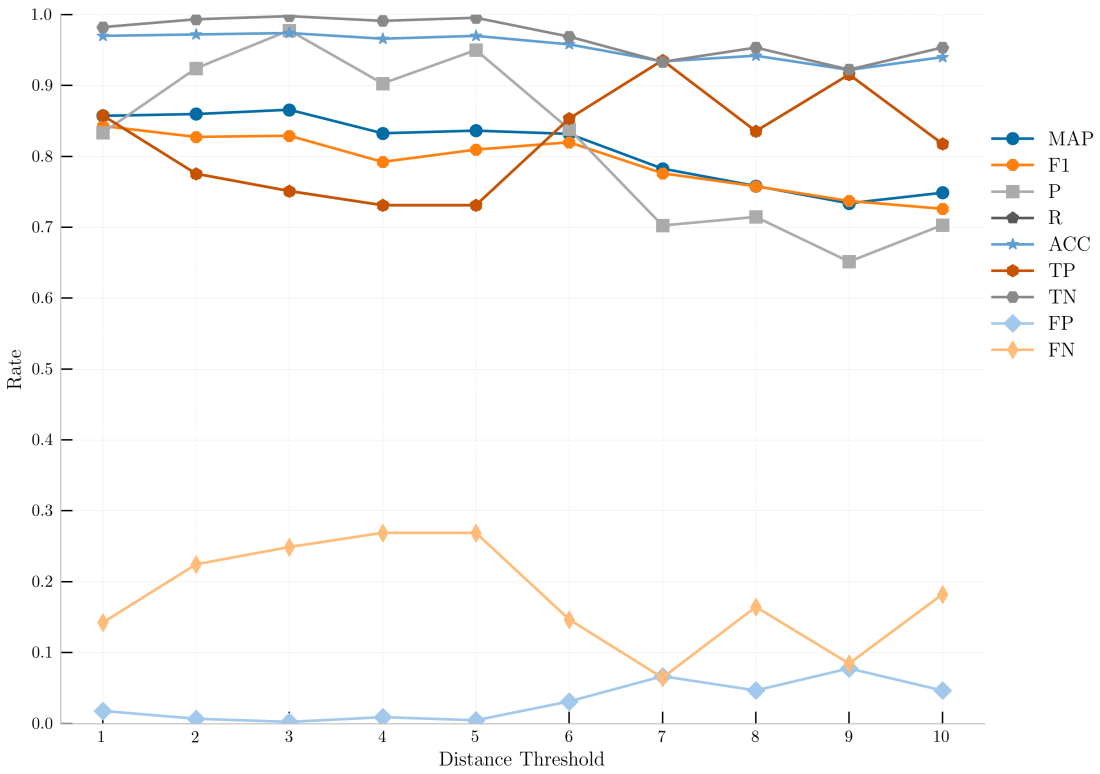


FIGURE 5.9: Evaluation metrics plots from Table 5.2

conducted on different thresholds. Table 5.2 reports the evaluation metrics corresponding $D \in \{1,\ldots,10\}$. The corresponding values are plotted in Figure 5.9. Figure 5.10 shows the precision-recall curves of examined thresholds.

5.5.2 HED Matching Parameters

HED matching requires four parameters $\{\tau_n, \tau_e, \alpha, \beta\}$ to be adjusted. τ_n and τ_e are the constant cost of deletion or insertion of nodes and edges. Based on the graph labels, we can determine which ranges of these parameters are more likely to be investigated. Small values of these parameters result in a HED matching with only deletion and insertion which holds no useful information. The parameter settings with large values might force the substructures to be substituted with dissimilar ones. Therefore, they must be selected to make a balance between mapping. We examine the range of $\tau_n, \tau_n \in \{1,2,4,8,16,32\}$

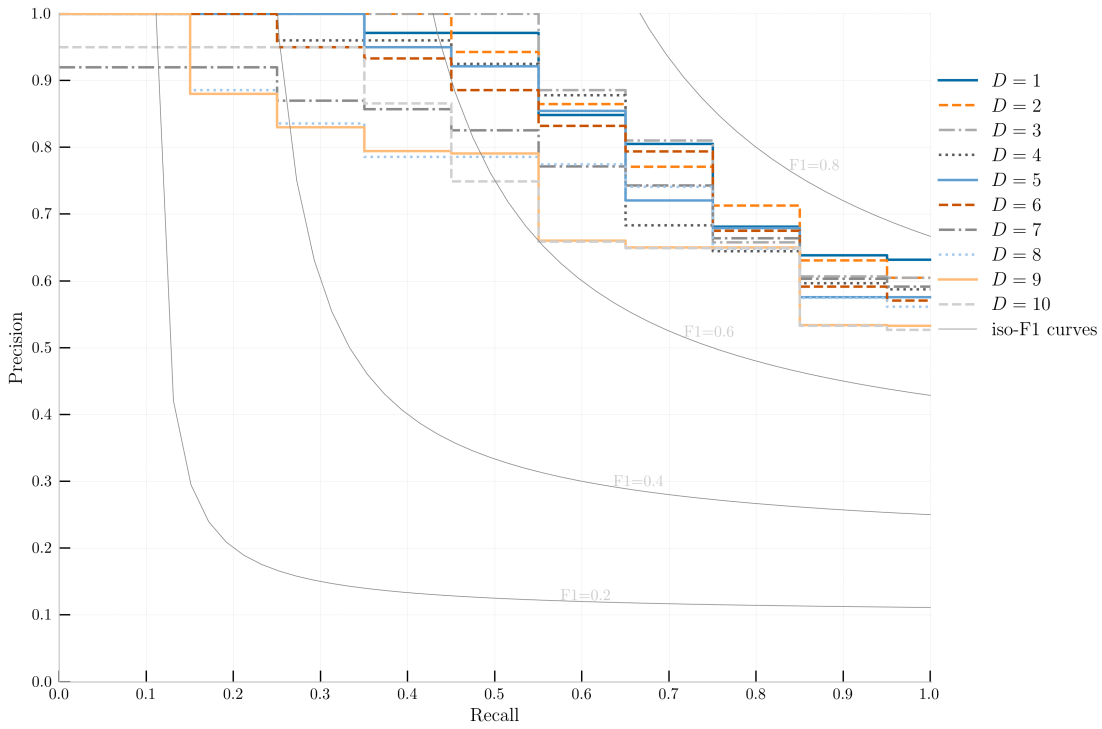


FIGURE 5.10: Precision-Recall metrics for connection points selection

and $\{0.1, 0.3, 0.5, 0.6, 0.7, 0.9\}$ for α and β parameters in the following experiments. The combination of parameters results in 900 experiments.

Best Parameters

In Table 5.3 the evaluation results from the best twenty parameters are provided. The results are sorted based on the MAP values. To give the reader the possibility to investigate further the results, we provide F1, ACC, P, R, TP, TN, FP, and FN metrics. The best ten parameters of Table 5.3 are plotted in Figure 5.11. Figure 5.12 illustrates the precision-recall curves of the best ten parameters and corresponding parameters values are annotated on the upper right side of the figure.

TABLE 5.3: Evaluation metrics for best twenty parameters

τ_n	τ_e	α	β	MAP	F1	ACC	P	R	TP	TN	FP	FN
2	2	0.3	0.1	95.85	94.60	99.00	98.00	91.78	91.78	99.78	0.22	8.22
2	1	0.3	0.1	95.82	94.22	99.00	100.00	89.28	89.28	100.00	0.00	10.72
1	4	0.5	0.1	95.66	92.57	98.60	94.14	91.78	91.78	99.34	0.66	8.22
1	2	0.3	0.1	95.54	93.42	98.80	95.78	91.78	91.78	99.56	0.44	8.22
4	2	0.3	0.3	95.53	94.37	99.00	97.78	91.78	91.78	99.78	0.22	8.22
1	16	0.7	0.1	95.53	93.28	98.80	98.00	89.56	89.56	99.78	0.22	10.44
2	4	0.5	0.1	95.49	92.38	98.60	93.56	91.78	91.78	99.34	0.66	8.22
2	2	0.3	0.3	95.45	93.33	98.80	96.00	91.78	91.78	99.57	0.43	8.22
2	16	0.7	0.1	95.44	93.28	98.80	98.00	89.56	89.56	99.78	0.22	10.44
2	16	0.7	0.3	95.39	92.10	98.60	95.78	89.56	89.56	99.56	0.44	10.44
2	8	0.5	0.1	95.37	92.42	98.60	96.36	89.56	89.56	99.56	0.44	10.44
2	2	0.5	0.1	95.34	92.28	98.60	93.78	91.78	91.78	99.35	0.65	8.22
4	16	0.7	0.3	95.34	92.10	98.60	95.78	89.56	89.56	99.56	0.44	10.44
4	1	0.1	0.1	95.31	92.12	98.60	97.78	87.56	87.56	99.78	0.22	12.44
4	8	0.5	0.1	95.29	92.70	98.60	94.14	91.78	91.78	99.34	0.66	8.22
2	8	0.7	0.1	95.26	91.88	98.60	97.78	87.56	87.56	99.78	0.22	12.44
2	32	0.9	0.1	95.26	91.88	98.60	97.78	87.56	87.56	99.78	0.22	12.44
8	2	0.5	0.1	95.25	94.23	99.00	100.00	89.56	89.56	100.00	0.00	10.44
2	1	0.1	0.1	95.25	92.12	98.60	97.78	87.56	87.56	99.78	0.22	12.44

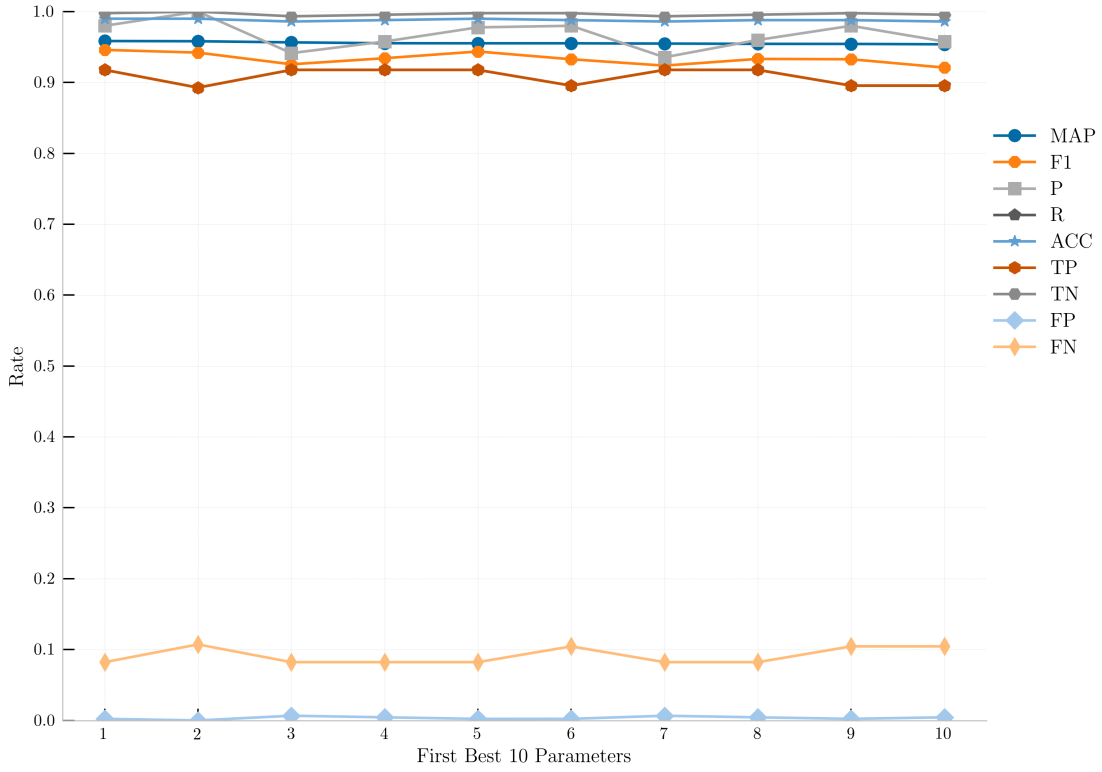


FIGURE 5.11: Evaluation metrics of best ten parameters from Table 5.3

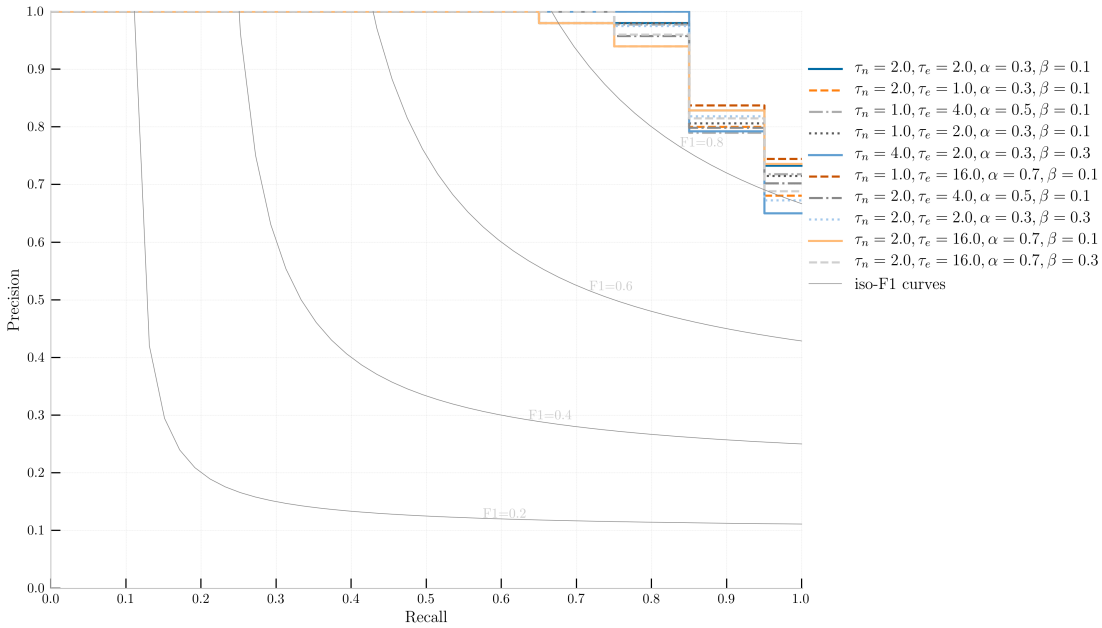


FIGURE 5.12: Precision-Recall curves for first best ten parameters

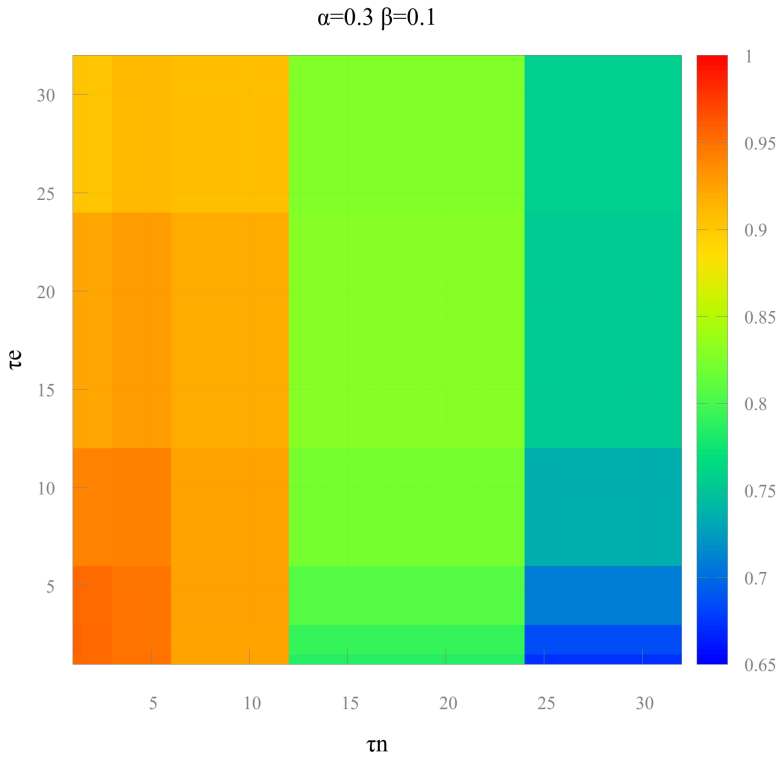


FIGURE 5.13: Variation of τ_n and τ_e parameters for $\alpha = 0.3$ and $\beta = 0.1$ in the parameter grid.

Performance Change on Parameter grids

Figure 5.13 illustrates the change of performance with respect to MAP values in a heat map plot. The values correspond to 36 pairs of $\tau_n \times \tau_e$ where $\alpha = 0.3$ and $\beta = 0.1$. We also fixed node and edge costs to $\tau_n = 2$ and $\tau_e = 2$ to obtain Figure 5.14 that demonstrates the performance change for the α and β parameters.

Five Queries Results for the Best Parameter

In the following, we provide the evaluation metrics and retrieval results of five queries using the best parameters $\tau_n = 2$, $\tau_e = 2$, $\alpha = 0.3$, and $\beta = 0.1$ in Table 5.4. Figure 5.15 shows performance metrics AP, F1, ACC, P, R, TP, TN, FP, and FN for five queries: *Colonel*, *now*, *no*, *October*, and *soon*. The first 20 retrieved words for query:*"October"* are provided in Figure 5.16. The images are labeled with the rank $= \{1,\ldots,20\}$ and HED

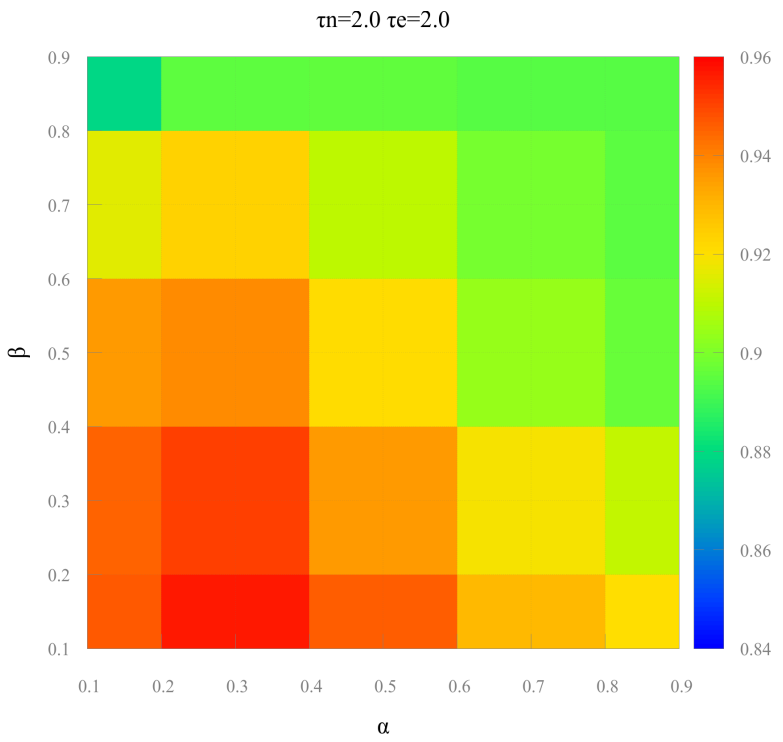


FIGURE 5.14: Variation of α and β parameters for $\tau_n = 2$ and $\tau_e = 2$ in the parameter grid.

distance. Figure 5.17 demonstrates the corresponding graph matchings of Figures 5.16. The graph matching figure contains ten rows. The first three rows are from rank=1-3 regardless of labels, and the rest are chosen from wrong matches. The exemplary spotting results show that the structural approach, indeed, finds keywords with a similar structure. Also, the top non-keywords have a close similarity to the query word, which is good. For instance, the word "shall" at rank=10 has visual similarity with "October", yet the substitutions have lengthier distances from the ones at the top-three ranks. In the deletion and insertion, we observe a more substantial number of nodes and edges that can be interpreted to more dissimilarities. Four other queries can be found in the appendix A in which we observe similar properties. The precision-recall, F1-recall and ROC curves are provided for the query in Figure 5.18.

TABLE 5.4: Evaluation metrics for five queries with the best parameter

Keyword	AP	F1	ACC	P	R	TP	TN	FP	FN
Colonel	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	0.00
now	92.88	90.00	98.00	90.00	90.00	90.00	98.89	1.11	10.00
no	95.56	94.12	99.00	100.00	88.89	88.89	100.00	0.00	11.11
October	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	0.00
soon	91.19	88.89	98.00	100.00	80.00	80.00	100.00	0.00	20.00

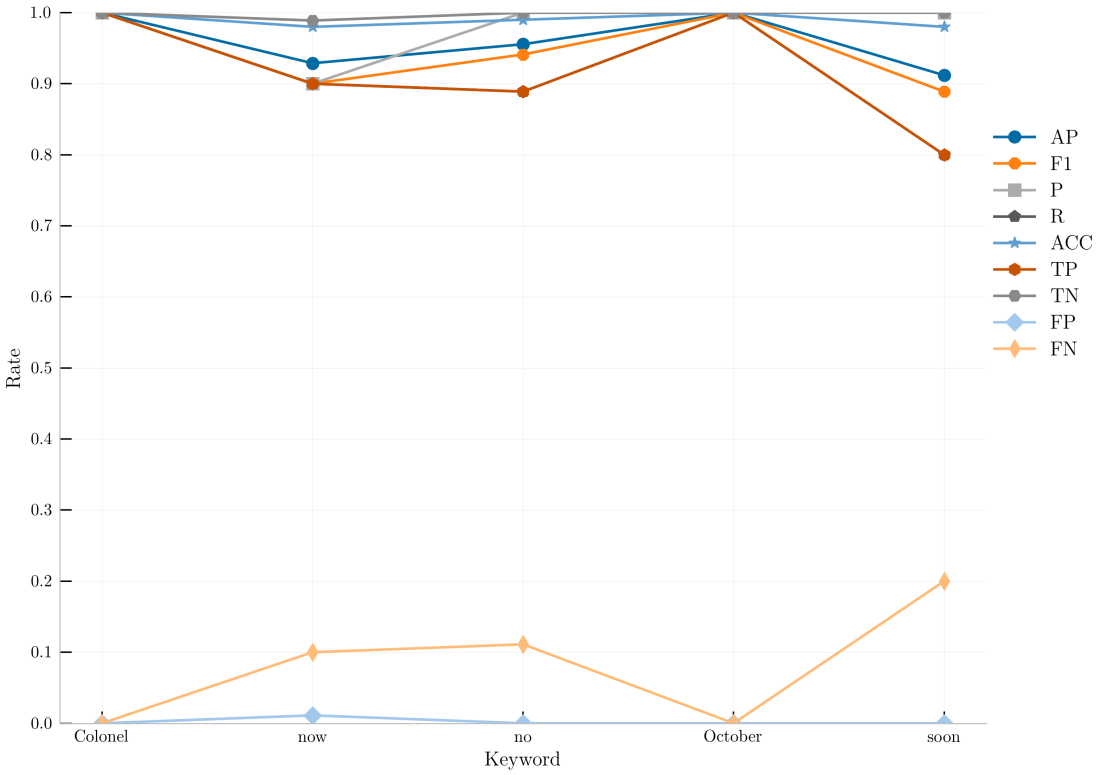


FIGURE 5.15: Evaluation metrics for five queries with the best parameters.

FIGURE 5.16: First 20 retrieved words for query *October*.

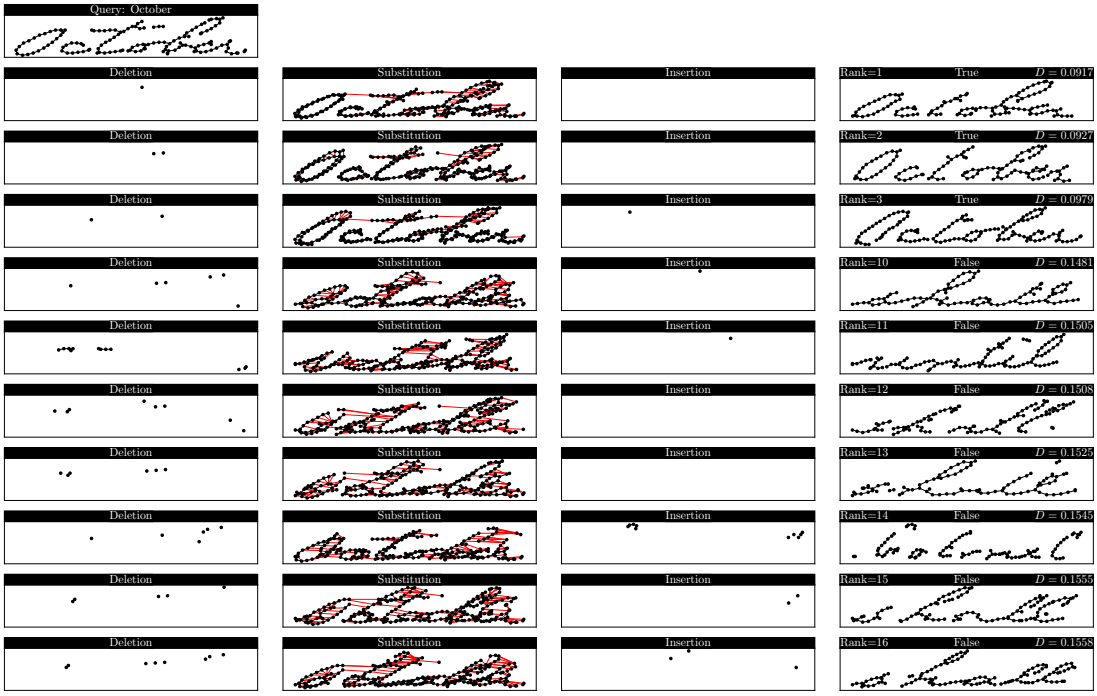


FIGURE 5.17: Ten samples from HED matching for query *October*.

5.5.3 Optimization of the Combined System

In this section, we verify using a single ω parameter in Eq. 5.7 for the combined HED-DTW system in which HED coefficient is set to one. By considering Eq. 5.11 instead of Eq. 5.7, we compute the MAP value for $(\omega_1, \omega_2) \in \{0.1, 0.2, \ldots, 5\} \times \{0.1, 0.2, \ldots, 5\}$ which constitute 2500 combinations of (ω_1, ω_2) in total.

$$r_{combined}(q,w) = \omega_1 \cdot r_{HED}(q,w) + \omega_2 \cdot r_{DTW}(q,w) \tag{5.11}$$

Figure 5.19 illustrates the heatmap plot of the (ω_1, ω_2) grid where the highest values are aligned along a straight line. Table 5.5 shows 20 of high performing parameter pairs with identical results. Therefore, we can fix one of the weights, for instance, ω_1 of HED and optimize the weight of DTW.

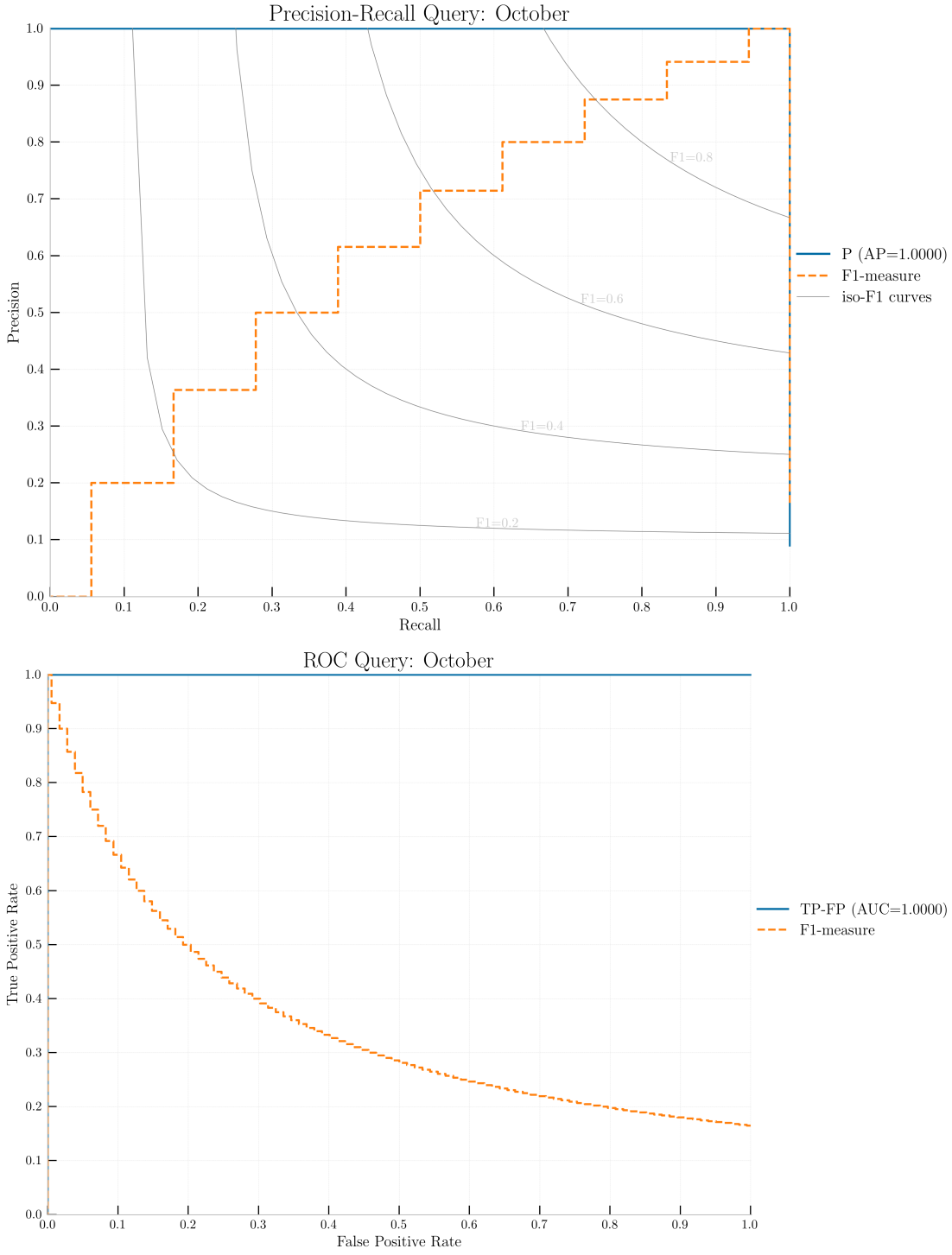


FIGURE 5.18: The precision-recall, F1-recall and ROC curves for query *October*.

TABLE 5.5: Combination of HED and DTW first 20 rows out of 55 similar results

HED	DTW	MAP	F1	ACC	P	R	TP	TN	FP	FN
4.9	1.3	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
0.7	0.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.6	0.9	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
1.4	0.4	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.9	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.8	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.7	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.6	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
3.5	1	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
2.4	0.6	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
2.9	0.8	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
2.8	0.8	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4.2	1.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4.8	1.3	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
0.8	0.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
1.2	0.3	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4.6	1.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4.5	1.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00
4.4	1.2	97.86	96.36	99.20	95.00	98.00	98.00	99.33	0.67	2.00

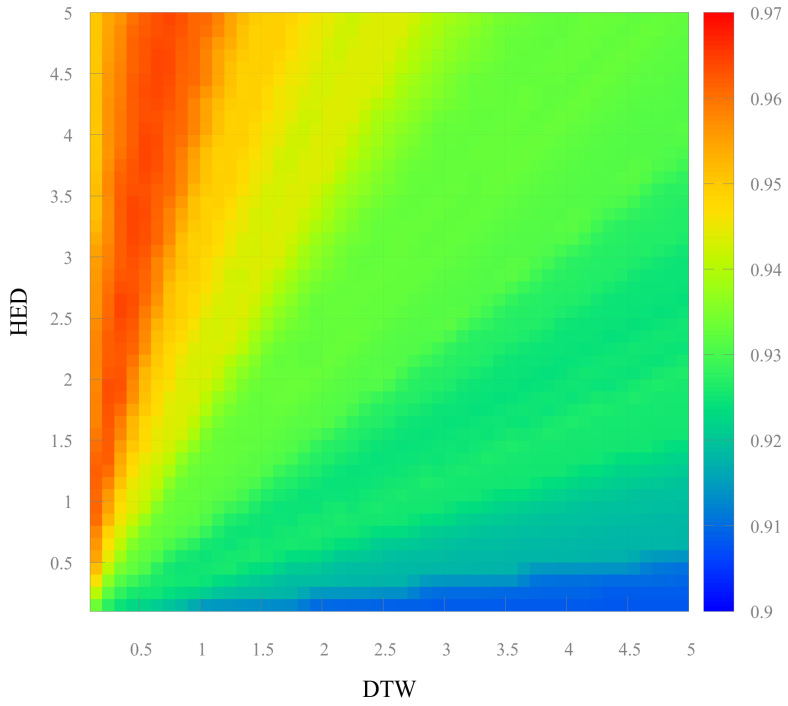


FIGURE 5.19: Relation between coefficients in combined HED-DTW KWS

5.6 Reference Methods

In order to assess the potential of the proposed HED-based graph matching approach, we compare it with three related reference methods. The reference systems include the methods using graphs matching (BP and BP2) and sequences alignment (DTW), respectively.

BP The first reference is the bipartite graph matching method (BP) proposed by Riesen and Bunke (2009a) for approximating the graph edit distance. BP is widely used for graph-based pattern recognition (see section 3.4). In particular, a number of graph-based keyword spotting systems, including Bui et al. (2015); Riba et al. (2015); Stauffer et al. (2016a); Stauffer, Fischer, and Riesen (2017); P. Wang et al. (2014b) employed the PB algorithm. BP reduces the problem of graph edit distance to an LSAP. Thus it returns a valid, yet not necessarily optimal, edit path between two graphs. BP approximates

the graph edit distance by computing an upper bound. The PB distance hence is used for computing the spotting score. The primary restriction of BP is still its cubic time complexity concerning the graph size. The time complexity is significantly improved in comparison with GED. However, it imposes computational limits regarding the size of the handwriting graphs as well as the number of handwriting graphs that can be matched.

BP2 The second reference is the recently introduced quadratic time variant of BP called BP2 (Fischer, Riesen, & Bunke, 2017). Similar to BP algorithm, BP2 solves the bipartite matching problem. However, it improves the cubic time complexity to quadratic time. The algorithm finds a valid, yet suboptimal edit path between two graphs thus an upper bound of graph edit distance.

DTW The third reference is based on the well-established Dynamic Time Warping (DTW) method for sequence matching. The approach has often been used for keyword spotting in historical manuscripts (Frinken et al., 2012; T. Rath & Manmatha, 2007; Terasawa & Tanaka, 2009; Wicht et al., 2016). The sequence of feature vectors is obtained by moving a sliding window over the handwriting. It is remarkable to note that the sequences are a particular case of graphs. The nodes are single feature vectors that have at most one successor. DTW finds an optimal alignment of two sequences, along with a time axis, in which the sum of feature vector distances is minimal. Using dynamic programming, an optimal DTW alignment can be obtained in quadratic time with respect to the sequence length. This sum of distances can then be used to compute a keyword spotting score (see section 2.1).

5.7 Experiments

The handwriting graphs are generated according to section 4.1. On all benchmark datasets, documents are segmented into individual word images. Then the segmentation results are manually corrected to exclude the segmentation mistakes in KWS experiments. Therefore, the evaluation has been performed on the accurately segmented words. In a real-world application, however, automatic word segmentation can contribute to decreasing the end-to-end performance.

We managed the experiments on the basis of two stages: validation and test. In the validation stage, we fine-tune the KWS system parameters. For each dataset, on a small validation set, the best parameters are retrieved. The validation set contains 10 random instances of 10 manually selected keywords (with different word lengths) and 900 additional, randomly selected words (1000 words in total) for GW and PAR datasets. For BOT and AK, we used the separate sets provided by the dataset. The training state evaluates the optimized system on the same training and test sets as used in Fischer et al. (2012) for GW and PAR and in Pratikakis et al. (2016) for AK and BOT. In Table 5.6 a summary of the datasets is presented.

TABLE 5.6: Number of keywords and number of word images in the training and test sets of the four datasets.

Dataset	Keywords	Train	Test
GW	105	2447	1224
PAR	1217	11468	6869
BOT	150	1684	3380
AK	200	1849	3734

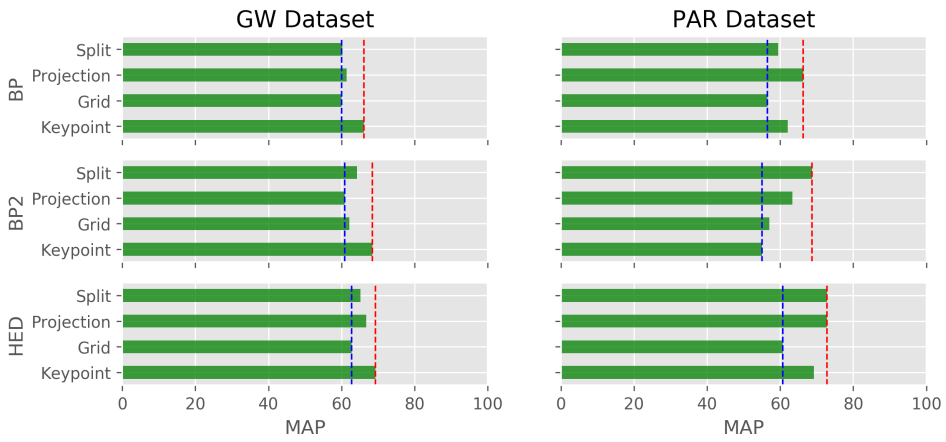


FIGURE 5.20: MAP values of Table 5.7

5.7.1 Comparison with Graph Edit Distance Approximations

The first experiment contains the evaluation of HED-based KWS compared to the other GED approximation approaches. BP and BP2, using a bipartite matching, approximate the graph edit distance. Similar to HED, BP and BP2 can be applied to any graph type without constraints on the graph structure or the node and edge label alphabets.

The parameter set includes the cost for node deletion/insertion τ_n, the cost for edge deletion/insertion τ_e, and the weights α, β of the cost function. They are optimized over the range of $\tau_n, \tau_e \in \{1,4,8,16,32\}$ and $\alpha, \beta \in \{0.1,0.3,0.5,0.7,0.9\}$ for each method individually on the validation set.

Table 5.7 presents the MAP results on the test set of GW and PAR for the three methods and the four graph representations. Figure 5.20 shows the plotted data of results in Table 5.7.

Comparing BP and BP2, we observe BP2 performs very similar to BP. The quadratic-time BP2 outperforms BP in five out of eight cases. Hence the BP2 is not only significantly more efficient concerning the time complexity but it can also achieve similar performance.

Our HED-based method achieves the best results, outperforming BP in eight out of

TABLE 5.7: Mean average precision (MAP) for graph-based KWS systems on the George Washington (GW) and Parzival (PAR) datasets.

		GW		PAR	
	Method	MAP	±	MAP	±
BP	Keypoint	66.08		62.04	
	Grid	60.02		56.50	
	Projection	61.43		66.23	
	Split	60.23		59.44	
BP2	Keypoint	68.42	+2.33	55.03	−7.01
	Grid	62.10	+2.07	57.00	+0.50
	Projection	60.83	−0.60	63.35	−2.88
	Split	64.24	+4.02	68.69	+9.25
HED	Keypoint	69.28	+3.19	69.23	+7.19
	Grid	62.78	+2.75	60.74	+4.24
	Projection	66.71	+5.28	72.82	+6.59
	Split	65.12	+4.89	72.79	+13.35

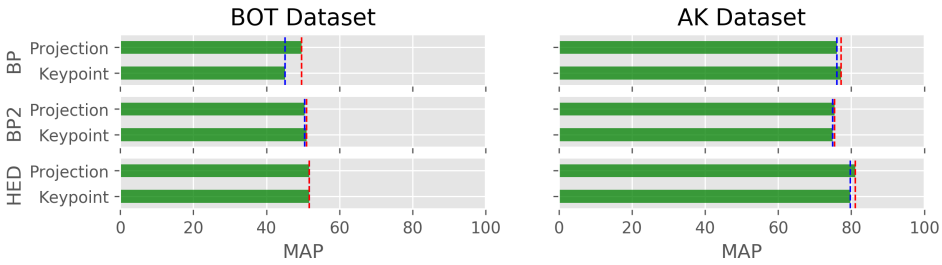


FIGURE 5.21: MAP values of Table 5.8

eight cases. Hence, it not only allows to reduce the computational complexity but also improves the keyword spotting performance. Unlike BP and BP2, HED allows multiple assignments among substructures in the handwriting graphs. Moreover, the edit path is not necessarily symmetric that provides a higher degree of flexibility. We assume that these properties of HED are beneficial in the context of handwriting. These allow handling minor distortion such as regarding the characters of different sizes. The DTW employs the same concept of *warping* only in one dimension rather than two.

The results, shown in Table 5.8, indicate a similar conclusion for the two other datasets, BOT and AK. Figure 5.21 plots MAP values of Table 5.8.

They confirm, with comparable findings to Table 5.7, that HED outperforms BP in four out of four cases on these datasets.

TABLE 5.8: Mean average precision (MAP) for graph-based KWS systems on the Botany (BOT) and Alvermann Konzilsprotokolle (AK) datasets.

		BOT		AK	
	Method	MAP	±	MAP	±
BP	`Keypoint`	45.06		77.24	
	`Projection`	49.57		76.02	
BP2	`Keypoint`	50.94	+5.88	74.86	−2.38
	`Projection`	50.49	+0.92	75.46	−0.56
HED	`Keypoint`	51.74	+6.68	79.72	+2.48
	`Projection`	51.69	+2.12	81.06	+5.04

Both HED and BP algorithms have significant speedup with polynomial time complexity. However, a cubic algorithm imposes a sizable delay to the experiments. Table 5.9 reports the speedup that can be achieved with the quadratic-time HED method when compared to the cubic-time BP method. On the GW dataset, the handwriting graphs have a median size between 74 and 90 and a maximum size between 366 and 509. For this graph size, HED-based keyword spotting is about hundred times faster than BP-based keyword spotting.

TABLE 5.9: Median and maximum number of nodes, mean runtime per graph pair in milliseconds for HED, BP, and BP2 with speed difference factor on the George Washington (GW) dataset.

			HED	BP		BP2	
Method	$\|V\|_{med}$	$\|V\|_{max}$	T	T	±	T	±
`Keypoint`	74	366	6.27	196.65	+190.38	6.73	+0.46
`Grid`	90	509	8.25	362.15	+353.89	8.18	-0.07
`Projection`	74	391	5.15	142.10	+136.96	5.78	+0.63
`Split`	80	434	5.83	136.24	+130.40	5.82	-0.01

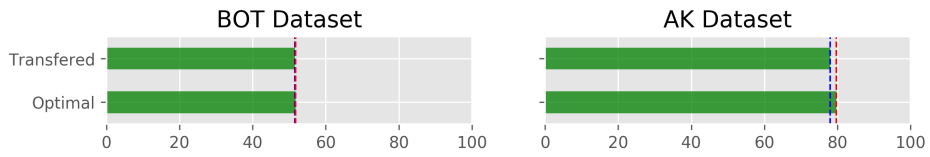


FIGURE 5.22: MAP values of Table 5.11.

Parameter Transfer

In this section, we present the outcome of experiments which are conducted with the transferred parameters from section 5.5.2. The transferred parameters are optimized on GW dataset and have been tested on BOT and AK datasets as shown in Table 5.10.

The following experiments show that the proposed system can achieve promising results on predefined parameters. The performance of the system slightly drops as indicated in Table 5.11 and corresponding plots in Figure 5.22.

TABLE 5.10: Optimal and transferred parameters.

	τ_n	τ_e	α	β
Transferred GW	2	2	0.3	0.1
Optimal AK	4	16	0.5	0.1
Optimal BOT	16	16	0.5	0.1

TABLE 5.11: Mean average precision (MAP) for HED-based KWS systems on the Botany (BOT) and Alvermann Konzilsprotokolle (AK) datasets with optimal and transferred parameters.

	BOT		AK	
Method	MAP	±	MAP	±
Optimal	51.74		79.72	
Transfered	51.59	-0.15	78.03	-1.69

5.7.2 Comparison with Dynamic Time Warping

In this section, we compare the graph-based KWS with state of the art template-based keyword spotting using DTW alignment. Three reference methods are considered for the GW and PAR benchmark datasets. The first approach, DTW'08 (Rodriguez & Perronnin, 2008) employs gradient features *histogram of oriented gradient (HoG)*. For extracting HoG features, the images are split into $M \times N$ cells. Then for each cell, horizontal and vertical gradients are used to calculate the polar gradient pairs (m, θ). The radial histogram T is then calculated by assigning the angles θ to the nearest bin. The sum of m in that specific bin constitutes the magnitude of bin $t \in T$. The overall number of features are therefore $M \times N \times T$. The second reference, the slit style feature, DTW'09 (Terasawa & Tanaka, 2009) performs the same task but on overlapping windows rather than segmented cells.

The third approach, DTW'16 (Wicht et al., 2016) is based on a convolutional neural network (CNN) features. The feature vectors are extracted from the datasets without supervision (without labeled training data) using deep belief networks. The approach stacks two *convolutional restricted Boltzmann machines (CRBM)*. The first layer is trained with extracted image patches. Then, the second layer is trained with the frozen weights from the first layer.

Table 5.12 shows a comparison with state of the art for template-based keyword spotting using DTW alignment. The DTW'09, DWT'08, and DTW'16 results are taken from Wicht et al. (2016), whereas, for HED, we show the results for the best performing graph representations found in Table 5.7. Figure 5.23 shows the plot of Table 5.12 values.

The results indicate that the template-based keyword spotting methods achieve performance results in the equivalent rate. DTW'09 and DTW'16 tend to outperform BP and BP2, while HED achieves the overall best results on these benchmarks.

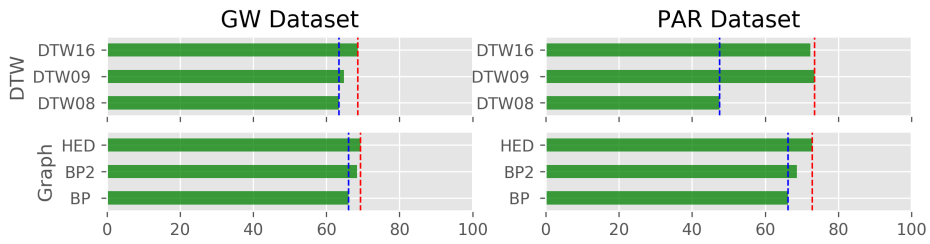


FIGURE 5.23: MAP values of Table 5.12

The strong performance of HED is somewhat surprising when comparing the sophisticated CNN features of DTW'16 with the relatively simple coordinate labels used for the handwriting graphs. It emphasizes the representational power of graphs for capturing relevant structures of the handwriting.

Regarding runtime, HED has a quadratic time complexity to the graph size and DTW has a quadratic time complexity concerning the sequence length. In our experimental setting, the graph size is typically smaller than the sequence length. On the GW dataset, for example, the median graph size is 74, while the median sequence length is 134. In this scenario, HED also reduces the computational effort when compared with DTW.

TABLE 5.12: Mean average precision (MAP) for graph-based KWS systems in comparison with three template-based reference systems on the George Washington (GW) and Parzival (PAR) dataset. The first, second, and third best systems are indicated by (1), (2), and (3).

Method		GW		PAR		Average	
Reference (Template)	DTW'08	63.39		47.52		55.46	
	DTW'09	64.80		73.49	(1)	69.15	(3)
	DTW'16	68.64	(2)	72.38	(3)	70.51	(2)
Graph (Template)	BP	66.08		66.23		66.16	
	BP2	68.42	(3)	68.69		68.55	
	HED	69.28	(1)	72.82	(2)	71.05	(1)

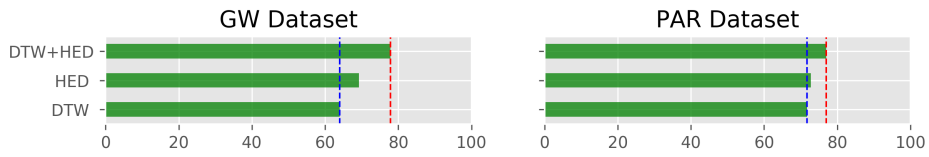


FIGURE 5.24: MAP values of Table 5.13

5.7.3 Combination of HED and Dynamic Time Warping

In the next experiment, we investigate the potential of combining HED and DTW proposed in section 5.2. We have implemented our DTW reference method, following the general ideas of T. Rath and Manmatha (2007) and using the features proposed by Marti and Bunke (2002). Image preprocessing includes skew and slant correction as well as height and width normalization. Afterward, a sliding window of one-pixel width extracts a sequence of nine geometric features. They are aligned utilizing DTW using a Sakoe-Chiba band (Sakoe & Chiba, 1978) with a width of Ω percent. The Sakoe-Chiba band speedups the alignment by excluding unusual warping paths. The parameter Ω is optimized on the validation set over a range of $\Omega \in \{0.20, 0.25, \ldots, 0.70\}$. The resulting cost of the warping path is normalized with the length of the warping path to obtain a keyword spotting score.

The normalized HED and the DTW scores are combined with a weighted sum $HED + \omega \cdot DTW$. The weight ω is optimized on the validation set over a range of $\omega \in \{0.1, 0.2, \ldots, 2.0\}$.

Table 5.13 reports the combination result on the GW and PAR test sets. The accompanying plots in Figure 5.24 illustrate the results.

The DTW system achieves a MAP of 64.00 on GW and 71.74 on PAR, which is comparable with the other reference methods listed in Table 5.12. Although DTW has a lower performance than HED, the combination leads to a significant increase in the MAP by

8.55% on GW and 4.18% on PAR. The results highlight the complementary properties of the two methods.

TABLE 5.13: Mean average precision (MAP) for the combination of DTW and HED on the George Washington (GW) and Parzival (PAR) datasets.

Method		GW		PAR	
Individual	DTW	64.00		71.74	
	HED	69.28		72.82	
Combined	DTW+HED	77.83	+8.55	77.00	+4.18

5.7.4 Comparison with Learning-Based Keyword Spotting

Our graph-based approach serves as a template-based keyword spotting approach . A template-based keyword spotting requires minimum human interaction in preparation and annotation of the collection. As we explained, it can search for the keyword in a collection of scanned documents with a single template image of the keyword. The low requirements of template-based keyword spotting are especially useful in the context of historical manuscripts, where obtaining labeled training data often requires human experts and thus becomes time-consuming and costly.

However, if labeled training data can be made available to the system, learning-based approaches can profit from this knowledge and build more robust spotting systems.

In Table 5.14 and corresponding plots in Figure 5.25, we compare our proposed template-based method with recent learning-based methods from the ICFHR2016 competition (Pratikakis et al., 2016), viz. CVCDAG (Almazan et al., 2014), PRG (Sudholt & Fink, 2016), and QTOB. CVCDAG is based on Pyramidal Histogram Of Characters (PHOC) features in conjunction with an SVM, PRG is based on the same features in conjunction with a Convolutional Neural Network (CNN), called PHOCNet. PHOCNet utilizes a 19 layer CNNs which consists of 13 convolutional, three max-pooling, and three fully connected

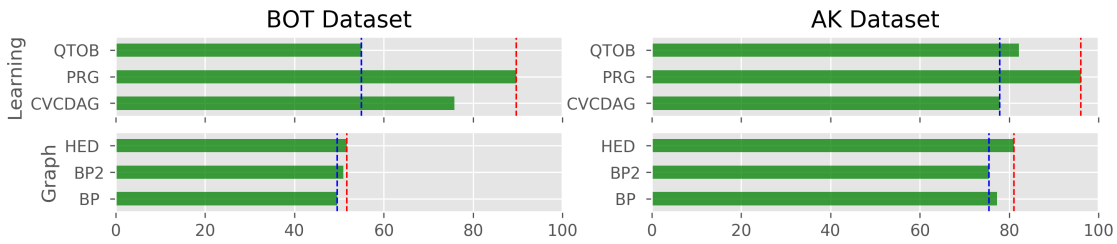


FIGURE 5.25: MAP values of Table 5.14

layers. QTOB is based on another CNN following a triplet network approach that uses 33 convolutional layers and one fully connected layer for the CNN architecture.

As expected, the learning-based methods achieve a higher performance in general, and especially PRG significantly outperforms the proposed HED-based method. Nevertheless, it is interesting to observe that HED can keep up with the performance of QTOB and outperforms CVCDAG in one out of three cases, despite the fact that no learning has been performed for HED. This observation demonstrates the high potential of HED as a template-based keyword spotting method.

Note that template-based and learning-based methods have complementary properties and can be used together in the digitization process of historical manuscripts. In the beginning, when no labeled data is available, a template-based method can be used to cluster similar words that are then labeled conjointly and efficiently by a human expert. As soon as enough training samples become available, learning-based methods can be trained to perform a more accurate search. Finally, when enough labeled data is available to train robust character models, a full transcription can be attempted together with a word dictionary (Frinken et al., 2014).

TABLE 5.14: Mean average precision (MAP) for graph-based KWS systems in comparison with three state-of-the-art learning-based reference systems on the Alvermann Konzilsprotokolle (AK) and Botany (BOT) datasets. The first, second, and third best systems are indicated by (1), (2), and (3).

Method		BOT		AK		Average	
Reference (Learning)	CVCDAG	75.77	(2)	77.91		76.84	(2)
	PRG	89.69	(1)	96.05	(1)	92.87	(1)
	QTOB	54.95	(3)	82.15	(2)	68.55	(3)
Graph (Template)	BP	49.57		77.24		63.41	
	BP2	50.94		75.46		63.20	
	HED	51.74		81.06	(3)	66.40	

5.8 Conclusion

The HED-based keyword spotting approach presented in this thesis has demonstrated several promising properties supported by empirical experiments. First, it approximates the graph edit distance and hence is *flexible* in the sense that it allows representing handwriting with any graph types, without constraints on the graph structure or the label alphabets for nodes and edges. Secondly, it can be computed in quadratic time concerning the graph size and hence is *efficient* for matching large graphs and large numbers of graphs. Thirdly, the experimental evaluation of system on four benchmark datasets for keyword spotting in historical manuscripts has demonstrated that it is *effective* in terms of mean average precision and compares favorably with other template-based keyword spotting systems.

Unlike dynamic time warping, which considers handwriting as a sequence of feature vectors, HED considers the two-dimensional global structure of the handwriting. The two perspectives are different and complementary. We have observed by combining the two methods into a multiple-classifier system that outperformed the individual methods.

Chapter 6

Conclusions and Outlook

In this thesis, we have proposed a graph-based keyword spotting system. The approach uses a flexible HED algorithm that can match any type of labeled graph. The computational complexity of HED is quadratic with respect to the number of graph nodes, which leads to a high keyword spotting efficiency similar to that of DTW. On four benchmark datasets, we have demonstrated that our proposed method is able to outperform other state-of-the-art template-based approaches, both in terms of accuracy and speed.

The statistical representation using feature vectors in DTW-based methods provides a different perspective on the handwriting when compared with the structural representation using graphs. By combining the two complementary methods, we have achieved a further significant improvement of the keyword spotting accuracy.

Handwritten keyword spotting remains an open field of research. We can suggest several promising lines of future research. In our opinion, the most compelling work would be a segmentation-free approach. One of the problems we have mentioned was the automatic segmentation and its induced error to the end-to-end keyword spotting system. We observed that the automatic segmentation may contribute to undesirable errors. We speculate that it might be rewarding to use graph-based representations for the whole page. In this scenario, keyword spotting would have a straightforward workflow by

eliminating or even reducing the complexity of segmentation. The segmentation-free approach requires matching a query to a whole document page. Hence the query must be regarded as a subset of a document in contrast to the one-to-one association. For instance, the page can contain several instances of the query keyword. The other concern would be the graph labels as they are coordinates in the image. These labels must accordingly be adjusted for the query or the documents.

Other future directions include the improvement of the current system such as an investigation of other, potentially more abstract graph-based representations of handwriting. It also may be rewarding to combine the HED spotting scores of different graph-based handwriting representations to improve the performance of the spotting system. Finally, given labeled training data is available, an intriguing open question is how to perform machine learning on graph-based representations and graph matching in order to profit from the labeled data.

Bibliography

Adamek, T., O'Connor, N. E., & Smeaton, A. F. (2006, jul). Word matching using single closed contours for indexing handwritten historical documents. *International Journal of Document Analysis and Recognition*, *9*(2-4), 153–165. Retrieved from http://link.springer.com/10.1007/s10032-006-0024-y doi: 10.1007/s10032-006-0024-y

Agazzi, O. (1994). Keyword spotting in poorly printed documents using pseudo 2-D hidden Markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(8), 842–848. Retrieved from http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=308482 doi: 10.1109/34.308482

Almazán, J., Gordo, A., Fornés, A., & Valveny, E. (2014, dec). Segmentation-free word spotting with exemplar SVMs. *Pattern Recognition*, *47*(12), 3967–3978. Retrieved from http://www.sciencedirect.com/science/article/pii/S0031320314002271 doi: 10.1016/j.patcog.2014.06.005

Almazan, J., Gordo, A., Fornes, A., & Valveny, E. (2014, dec). Word Spotting and Recognition with Embedded Attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(12), 2552–2566.

Ameri, M. R., Haji, M., Fischer, A., Ponson, D., & Bui, T. D. (2014, Sept). A feature extraction method for cursive character recognition using higher-order singular value decomposition. In *2014 14th international conference on frontiers in handwriting recognition* (p. 512-516).

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T., & Fischer, A. (2017). Keyword

Spotting in Historical Documents Based on Handwriting Graphs and Hausdorff Edit Distance. In *International graphonomics society conference* (pp. 105–108).

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T. D., & Fischer, A. (2018). Graph-based keyword spotting in historical manuscripts using hausdorff edit distance. *Pattern Recognition Letters*.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., & Kriegel, H.-P. (2005, January). Protein function prediction via graph kernels. *Bioinformatics*, *21*(1), 47–56. Retrieved from `http://dx.doi.org/10.1093/bioinformatics/bti1007` doi: 10.1093/bioinformatics/bti1007

Bui, Q. A., Visani, M., & Mullot, R. (2015). Unsupervised word spotting using a graph representation based on invariants. In *International conference on document analysis and recognition* (pp. 616–620).

Bunke, H. (1993). Structural and syntactic pattern recognition. In *Handbook of pattern recognition and computer vision* (pp. 163–209). World Scientific.

Bunke, H., & Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, *1*(4), 245–253.

Bunke, H., & Varga, T. (2007). Off-line Roman cursive handwriting recognition. In B. Chaudhuri (Ed.), *Digital document processing* (pp. 165–173). Springer.

Burkard, R., Dell'Amico, M., & Martello, S. (2009). *Assignment Problems*. Society for Industrial and Applied Mathematics.

Can, E. F., & Duygulu, P. (2011). *A line-based representation for matching words in historical manuscripts* (Vol. 32) (No. 8). Retrieved from `http://www.sciencedirect.com/science/article/pii/S016786551100050X` doi: 10.1016/j.patrec.2011.02.013

Cao, H., & Govindaraju, V. (2007). Template-free word spotting in low-quality manuscripts. In *International conference on advances in pattern recognition* (pp. 1–5).

Cavalin, P. R., Sabourin, R., Suen, C. Y., & Britto Jr., A. S. (2009, December). Evaluation of incremental learning algorithms for hmm in the recognition of alphanumeric characters. *Pattern Recogn.*, *42*(12), 3241–3253. Retrieved from http://dx.doi.org/10.1016/j.patcog.2008.10.012 doi: 10.1016/j.patcog.2008.10.012

Chan, J., Ziftci, C., & Forsyth, D. (2006). Searching off-line arabic documents. In *Ieee computer society conference on computer vision and pattern recognition* (Vol. 2, pp. 1455–1462). doi: 10.1109/CVPR.2006.269

Chen, G., Bui, T., & Krzyzak, A. (2003). Contour-based handwritten numeral recognition using multiwavelets and neural networks. *Pattern Recognition*, *36*(7), 1597–1604.

Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(03), 265–298.

Dey, S., Nicolaou, A., Llados, J., & Pal, U. (2016). Local Binary Pattern for Word Spotting in Handwritten Historical Document. In *International workshop on structural, syntactic, and statistical pattern recognition* (pp. 574–583). Retrieved from http://arxiv.org/abs/1604.05907http://link.springer.com/10.1007/978-3-319-49055-7{_}51 doi: 10.1007/978-3-319-49055-7_51

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification (2nd edition)*. Wiley-Interscience.

Edwards, J., Teh, Y. W., Bock, R., Maire, M., Vesom, G., & Forsyth, D. A. (2004). Making latin manuscripts searchable using gHMM's. In *International conference on neural information processing systems* (Vol. 17, pp. 385–392).

Escalera, S., Fornés, A., Pujol, O., Radeva, P., Sánchez, G., & Lladós, J. (2009). Blurred shape model for binary and grey-level symbol recognition. *Pattern Recognition Letters*, *30*(15), 1424 - 1433. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167865509002074 doi: https://doi.org/10.1016/

j.patrec.2009.08.001

Fischer, A., & Bunke, H. (2009). Kernel PCA for HMM-based cursive handwriting recognition. In *Proc. 13th int. conf. on computer analysis of images and patterns* (pp. 181–188).

Fischer, A., Indermühle, E., Bunke, H., Viehhauser, G., & Stolz, M. (2010). Ground truth creation for handwriting recognition in historical documents. In *International workshop on document analysis systems* (pp. 3–10). New York, New York, USA.

Fischer, A., Keller, A., Frinken, V., & Bunke, H. (2012). Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*, *33*(7), 934 - 942. (Special Issue on Awards from ICPR 2010)

Fischer, A., Plamondon, R., Savaria, Y., Riesen, K., & Bunke, H. (2014). A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance. In *International workshop on structural, syntactic, and statistical pattern recognition* (Vol. 8621, pp. 83–92). Springer Berlin Heidelberg. Retrieved from `http://link.springer.com/10.1007/978-3-662-44415-3{_}9` doi: 10.1007/978-3-662-44415-3_9

Fischer, A., Riesen, K., & Bunke, H. (2010, Nov). Graph similarity features for HMM-based handwriting recognition in historical documents. In *Frontiers in handwriting recognition (icfhr), 2010 international conference on* (p. 253-258).

Fischer, A., Riesen, K., & Bunke, H. (2017). Improved quadratic time approximation of graph edit distance by combining Hausdorff matching and greedy assignment. *Pattern Recognition Letters*, *87*, 55–62.

Fischer, A., Suen, C. Y., Frinken, V., Riesen, K., & Bunke, H. (2013). A fast matching algorithm for graph-based handwriting recognition. In *Graph-based representations in pattern recognition* (Vol. 7877, pp. 194–203). doi: 10.1007/978-3-642-38221-5_21

Fischer, A., Suen, C. Y., Frinken, V., Riesen, K., & Bunke, H. (2015, feb). Approximation of graph edit distance based on Hausdorff matching. *Pattern Recognition*, *48*(2),

331–343.

Foggia, P., Percannella, G., & Vento, M. (2014). Graph Matching and Learning in Pattern Recognition in the last 10 Years. *International Journal of Pattern Recognition and Artificial Intelligence*, *28*(01), 1450001.

Frinken, V., Fischer, A., Baumgartner, M., & Bunke, H. (2014). Keyword spotting for self-training of BLSTM NN based handwriting recognition systems. In *Pattern recognition* (Vol. 47, pp. 1073–1082).

Frinken, V., Fischer, A., Manmatha, R., & Bunke, H. (2012). A novel word spotting method based on recurrent neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(2), 211–224.

Fujisawa, H. (2008). Forty years of research in character and document recognition an industrial perspective. *Pattern Recognition*, *41*(8), 2435 - 2446. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0031320308000964` doi: http://dx.doi.org/10.1016/j.patcog.2008.03.015

Gorski, N., Anisimov, V., Augustin, E., Baret, O., & Maximor, S. (2001). Industrial bank check processing: The A2iA check reader. *Int. Journal on Document Analysis and Recognition*, *3*, 196-206.

Grother, P. J. (1995). *NIST special database 19 - handprinted forms and characters database* (Tech. Rep.). National Institute of Standards and Thechnology (NIST).

Guo, Z., & Hall, R. W. (1989). Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, *32*(3), 359–373.

Haji, M. (2012). *Arbitrary keyword spotting in handwritten documents* (Unpublished doctoral dissertation). Concordia University.

Haji, M., Ameri, M. R., Bui, T. D., Suen, C. Y., & Ponson, D. (2014). Two-stage approach to keyword spotting in handwritten documents. In *Proceedings of spie - the international society for optical engineering* (Vol. 9021).

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4082128 doi: 10.1109/TSSC.1968.300136

Howe, N. R. (2013). Part-structured inkball models for one-shot handwritten word spotting. In *Proceedings of the international conference on document analysis and recognition, icdar* (pp. 582–586).

Hull, J. (1998). Document image skew detection: Survey and annotated bibliography. In *Series in machine perception and artificial intelligence* (Vol. 29, pp. 40–64). World Scientific. Retrieved from http://books.google.com/books?hl=en{&}lr={&}id=HRW8kk4e4{_}QC{&}oi=fnd{&}pg=PA40{&}dq=Document+image+skew+detection:+survey+and+annotated+bibliography{&}ots=kCatROWWE4{&}sig=mx2qgQPeiwJuq6CIQPzvrxv053o

Kandel, A., Bunke, H., & Last, M. (2007). *Applied graph theory in computer vision and pattern recognition* (Vol. 52). Springer.

Konidaris, T., Kesidis, A. L., & Gatos, B. (2015, may). A segmentation-free word spotting method for historical printed documents. *Pattern Analysis and Applications*. Retrieved from http://link.springer.com/10.1007/s10044-015-0476-0 doi: 10.1007/s10044-015-0476-0

Koopmans, T. C., & Beckmann, M. (1957, jan). Assignment Problems and the Location of Economic Activities. *Econometrica*, *25*(1), 53.

Kovalchuk, A., Wolf, L., & Dershowitz, N. (2014, sep). A Simple and Fast Word Spotting Method. In *International conference on frontiers in handwriting recognition* (pp. 3–8). IEEE. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6980988 doi: 10.1109/ICFHR.2014.9

Kruskal, J. B. (1956, jan). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, *7*(1),

48–48. Retrieved from http://www.ams.org/jourcgi/jour-getitem?pii=S0002-9939-1956-0078686-7 doi: 10.1090/S0002-9939-1956-0078686-7

Kuncheva, L. I. (2004). *Combining Pattern Classifiers*. Hoboken, NJ, USA: John Wiley & Sons, Inc.

Lavrenko, V., Rath, T., & Manmatha, R. (2004). Holistic word recognition for handwritten historical documents. In *International workshop on document image analysis for libraries* (pp. 278–287). IEEE. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1263256 doi: 10.1109/DIAL.2004.1263256

Leydier, Y., Lebourgeois, F., & Emptoz, H. (2007, dec). Text search for medieval manuscript images. *Pattern Recognition*, *40*(12), 3552–3567. Retrieved from http://www.sciencedirect.com/science/article/pii/S0031320307002208 doi: 10.1016/j.patcog.2007.04.024

Lu, Y., & Tan, C. L. (2002). Combination of multiple classifiers using probabilistic dictionary and its application to postcode recognition. *Pattern Recognition*, *35*(12), 2823 - 2832. Retrieved from http://www.sciencedirect.com/science/article/pii/S0031320301002370 (Pattern Recognition in Information Systems) doi: https://doi.org/10.1016/S0031-3203(01)00237-0

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., & Vert, J.-P. (2005). Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, *45*(4), 939–951. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/16045288 doi: 10.1021/ci050039t

Manmatha, R., & Croft, W. (1997). Word spotting: Indexing handwritten archives. In *Maybury, m.t. (ed.), intelligent multimedia information retrieval. mit press* (p. 43-64).

Manmatha, R., Han, C., & Riseman, E. M. (1996, Jun). Word spotting: a new approach to indexing handwriting. In *Computer vision and pattern recognition, 1996.*

proceedings cvpr '96, 1996 ieee computer society conference on (p. 631-637).

Manmatha, R., & Rath, T. M. (2003). Indexing of Handwritten Historical Documents - Recent Progress. *Symposium on Document Image Understanding Technology*, 77–85. Retrieved from `http://citeseer.ist.psu.edu/586294.html`;

Marti, U., & Bunke, H. (2002). Hidden markov models. In (pp. 65–90). River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Myers, C., Rabiner, L., & Rosenberg, A. (1980, Apr). An investigation of the use of dynamic time warping for word spotting and connected speech recognition. In *Acoustics, speech, and signal processing, ieee international conference on icassp '80.* (Vol. 5, p. 173-177).

National Institute of Standards and Technology. (2009). Common Evaluation Measures. *The Eighteenth Text REtrieval Conference (TREC 2009) Proceedings*. Retrieved from `http://trec.nist.gov/pubs/trec15/appendices/CE.MEASURES06.pdf`

Olszewski, R. (2001, 03). Generalized feature extraction for structural pattern recognition in time-series data.

Pekalska, E., & Duin, R. P. W. (2005). *The dissimilarity representation for pattern recognition: Foundations and applications (machine perception and artificial intelligence)*. River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Perronnin, F., & Rodríguez-Serrano, J. A. (2009). Fisher Kernels for Handwritten Word-spotting. In *International conference on document analysis and recognition* (pp. 106–110).

Pratikakis, I., Zagoris, K., Gatos, B., Puigcerver, J., Toselli, A. H., & Vidal, E. (2016, oct). ICFHR2016 Handwritten Keyword Spotting Competition (H-KWS 2016). In *International conference on frontiers in handwriting recognition* (pp. 613–618). IEEE.

Rabiner, L. (1989, Feb). A tutorial on hidden markov models and selected applications

in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, *18*(8), 1093 - 1110. Retrieved from http://www.sciencedirect.com/science/article/pii/S0893608005001693 (Neural Networks and Kernel Methods for Structured Domains) doi: https://doi.org/10.1016/j.neunet.2005.07.009

Rath, T., & Manmatha, R. (2003). Word image matching using dynamic time warping. In *Computer vision and pattern recognition* (Vol. 2, pp. II–521–II–527). IEEE. Retrieved from http://www.mendeley.com/catalog/word-image-matching-using-dynamic-time-warping/{%}5Cnhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1211511{%}5Cnhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1211511http://ieeexplore.ieee.org/lpdocs/epic doi: 10.1109/CVPR.2003.1211511

Rath, T., & Manmatha, R. (2007). Word spotting for historical documents. *International Journal of Document Analysis and Recognition (IJDAR)*, *9*(2-4), 139-152.

Rath, T. M., Manmatha, R., & Lavrenko, V. (2004). A search engine for historical manuscript images. In *Proceedings of the 27th annual international acm sigir conference on research and development in information retrieval* (pp. 369–376). New York, NY, USA: ACM.

Riba, P., Llados, J., & Fornes, A. (2015). Handwritten word spotting by inexact matching of grapheme graphs. In *International conference on document analysis and recognition* (pp. 781–785).

Riesen, K. (2015). *Structural Pattern Recognition with Graph Edit Distance*. Cham: Springer International Publishing.

Riesen, K., Brodić, D., Milivojević, Z. N., & Maluckov, Č. A. (2014). Graph Based Keyword Spotting in Medieval Slavic Documents – A Project Outline.

In *International conference on cultural heritage* (pp. 724–731). Springer International Publishing. Retrieved from http://link.springer.com/10.1007/978-3-319-13695-0{_}74 doi: 10.1007/978-3-319-13695-0_74

Riesen, K., & Bunke, H. (2009a). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, *27*(7), 950–959.

Riesen, K., & Bunke, H. (2009b). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, *27*(7), 950 - 959. (7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007))

Riesen, K., Fischer, A., & Bunke, H. (2015, mar). Estimating Graph Edit Distance Using Lower and Upper Bounds of Bipartite Approximations. *International Journal of Pattern Recognition and Artificial Intelligence*, *29*(02), 1550011. Retrieved from http://www.worldscientific.com/doi/10.1142/S0218001415500111 doi: 10.1142/S0218001415500111

Riesen, K., Neuhaus, M., & Bunke, H. (2007). Bipartite Graph Matching for Computing the Edit Distance of Graphs. In *Graph-based representations in pattern recognition* (Vol. 6, pp. 1–12). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dl.acm.org/citation.cfm?id=1769371.1769373http://link.springer.com/10.1007/978-3-540-72903-7{_}1 doi: 10.1007/978-3-540-72903-7_1

Rocha, J., & Pavlidis, T. (1994, April). A shape analysis model with applications to a character recognition system. *IEEE Trans. Pattern Anal. Mach. Intell.*, *16*(4), 393–404. Retrieved from http://dx.doi.org/10.1109/34.277592 doi: 10.1109/34.277592

Rodriguez, J., & Perronnin, F. (2008, 8). Local gradient histogram features for word spotting in unconstrained handwritten documents. In *Icfhr 2008 (international conference on frontiers in handwriting recognition)* (p. 19-21).

Rodríguez-Serrano, J. A., & Perronnin, F. (2012, nov). A model-based sequence similarity

with application to handwritten word spotting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(11), 2108–20. Retrieved from `http://www.ncbi.nlm.nih.gov/pubmed/22248634` doi: 10.1109/TPAMI.2012.25

Rodríguez-Serrano, J. A., & Perronnin, F. (2009). Handwritten word-spotting using hidden markov models and universal vocabularies. *Pattern Recognition*, *42*(9), 2106 - 2116.

Rothacker, L., & Fink, G. A. (2015, aug). Segmentation-free query-by-string word spotting with Bag-of-Features HMMs. In *International conference on document analysis and recognition* (pp. 661–665). IEEE.

Rusiñol, M., Aldavert, D., Toledo, R., & Lladós, J. (2015). Efficient segmentation-free keyword spotting in historical document collections. *Pattern Recognition*, *48*(2), 545–555.

Sakoe, H., & Chiba, S. (1978, feb). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *26*(1), 43–49.

Sanfeliu, A., & Fu, K.-S. (1983, may). A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(3), 353–362. Retrieved from `http://ieeexplore.ieee.org/document/6313167/` doi: 10.1109/TSMC.1983.6313167

Sankaran, N., & Jawahar, C. (2012). Recognition of Printed Devanagari Text Using BLSTM Neural Network. In *International conference on pattern recognition* (pp. 322–325).

Schenker, A., Bunke, H., Last, M., & Kandel, A. (2005). *Graph-theoretic techniques for web content mining*. River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Schenker, A., Last, M., Bunke, H., & Kandel, A. (2004, 05). Classification of web documents using graph matching. , *18*, 475-496.

Schlapbach, A., & Bunke, H. (2008). Off-line writer identification and verification

using gaussian mixture models. In S. Marinai & H. Fujisawa (Eds.), *Machine learning in document analysis and recognition* (pp. 409–428). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-76280-5_16 doi: 10.1007/978-3-540-76280-5_16

Scott, G. L., & Longuet-Higgins, H. C. (1991, apr). An Algorithm for Associating the Features of Two Images. *Proceedings of the Royal Society B: Biological Sciences*, *244*(1309), 21–26.

Stauffer, M., Fischer, A., & Riesen, K. (2016a). Graph-based Keyword Spotting in Historical Handwritten Documents. In *International workshop on structural, syntactic, and statistical pattern recognition* (pp. 564–573).

Stauffer, M., Fischer, A., & Riesen, K. (2016b). A Novel Graph Database for Handwritten Word Images. In *International workshop on structural, syntactic, and statistical pattern recognition* (pp. 553–563).

Stauffer, M., Fischer, A., & Riesen, K. (2017). Ensembles for Graph-based Keyword Spotting in Historical Handwritten Documents. In *International conference on document analysis and recognition.*

Sudholt, S., & Fink, G. A. (2016, oct). PHOCNet: A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents. In *International conference on frontiers in handwriting recognition* (pp. 277–282). IEEE.

Suganthan, P., & Yan, H. (1998). Recognition of handprinted chinese characters by constrained graph matching. *Image and Vision Computing*, *16*(3), 191 - 201. Retrieved from http://www.sciencedirect.com/science/article/pii/S0262885697000668 doi: https://doi.org/10.1016/S0262-8856(97)00066-8

Terasawa, K., & Tanaka, Y. (2009). Slit Style HOG Feature for Document Image Word Spotting. In *International conference on document analysis and recognition* (pp. 116–120).

Trier, O. D., Jain, A. K., & Taxt, T. (1996). Feature extraction methods for character

recognition – A survey. *Pattern Recognition*, *29*(4), 641–662.

Vinciarelli, A., & Bengio, S. (2002). Off-line cursive word recognition using continuous density HMMs trained with PCA and ICA features. In *Proc. 16th int. conf. on pattern recognition* (Vol. 3, p. 81-84).

Wang, P., Eglin, V., Garcia, C., Largeron, C., Llados, J., & Fornes, A. (2014a). A Coarse-to-Fine Word Spotting Approach for Historical Handwritten Documents Based on Graph Embedding and Graph Edit Distance. In *International conference on pattern recognition* (pp. 3074–3079). IEEE.

Wang, P., Eglin, V., Garcia, C., Largeron, C., Llados, J., & Fornes, A. (2014b). A Novel Learning-Free Word Spotting Approach Based on Graph Representation. In *International workshop on document analysis systems* (pp. 207–211).

Wang, X., Ding, X., & Liu, C. (2005, March). Gabor filters-based feature extraction for character recognition. *Pattern Recogn.*, *38*(3), 369–379. Retrieved from `http://dx.doi.org/10.1016/j.patcog.2004.08.004` doi: 10.1016/j.patcog.2004.08.004

Wicht, B., Fischer, A., & Hennebert, J. (2016). Deep Learning Features for Handwritten Keyword Spotting. In *International conference on pattern recognition* (pp. 3423–3428).

Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T., & Yan, S. (2010). Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, *98*(6), 1031–1044.

Wunsch, P., & Laine, A. F. (1995). Wavelet descriptors for multiresolution recognition of handprinted characters. *Pattern Recognition*, *28*(8), 1237—1249.

Yager, N., & Amin, A. (2004, April). Fingerprint classification: A review. *Pattern Anal. Appl.*, *7*(1), 77–93. Retrieved from `http://dx.doi.org/10.1007/s10044-004-0204-7` doi: 10.1007/s10044-004-0204-7

Zhang, B., Srihari, S. N., & Huang, C. (2003, dec). Word image retrieval using binary

features. In E. H. Barney Smith, J. Hu, & J. Allan (Eds.), *Document recognition and retrieval* (p. 45). International Society for Optics and Photonics. Retrieved from http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=837077http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.523968 doi: 10.1117/12.523968

Zhang, L., Zhu, J., & Yao, T. (2004, December). An evaluation of statistical spam filtering techniques. , *3*(4), 243–269. Retrieved from http://doi.acm.org/10.1145/1039621.1039625 doi: 10.1145/1039621.1039625

Zimmermann, M., Chappelier, J.-C., & Bunke, H. (2006, May). Offline grammar-based recognition of handwritten sentences. *IEEE Trans. Pattern Anal. Mach. Intell.*, *28*(5), 818–821. Retrieved from http://dx.doi.org/10.1109/TPAMI.2006.103 doi: 10.1109/TPAMI.2006.103

Publications

Publications which are relevant to the thesis:

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T., & Fischer, A. (2017). Keyword Spotting in Historical Documents Based on Handwriting Graphs and Hausdorff Edit Distance. In *International graphonomics society conference* (pp. 105–108).

Ameri, M. R., Stauffer, M., Riesen, K., Bui, T. D., & Fischer, A. (2018). Graph-based keyword spotting in historical manuscripts using hausdorff edit distance. *Pattern Recognition Letters*.

Other publications during the PhD program:

Ameri, M. R., Haji, M., Fischer, A., Ponson, D., & Bui, T. D. (2014, Sept). A feature extraction method for cursive character recognition using higher-order singular value decomposition. In *2014 14th international conference on frontiers in handwriting recognition* (p. 512-516).

Haji, M., Ameri, M. R., Bui, T. D., Suen, C. Y., & Ponson, D. (2014). Two-stage approach to keyword spotting in handwritten documents. In *Proceedings of spie - the international society for optical engineering* (Vol. 9021).

Appendix A

Retrieval Output of Section 5.5.2

FIGURE A.1: First 20 retrieved words for query *Colonel*.

FIGURE A.2: First 20 retrieved words for query *no*.

FIGURE A.3: First 20 retrieved words for query *now*.

FIGURE A.4: First 20 retrieved words for query *soon*.

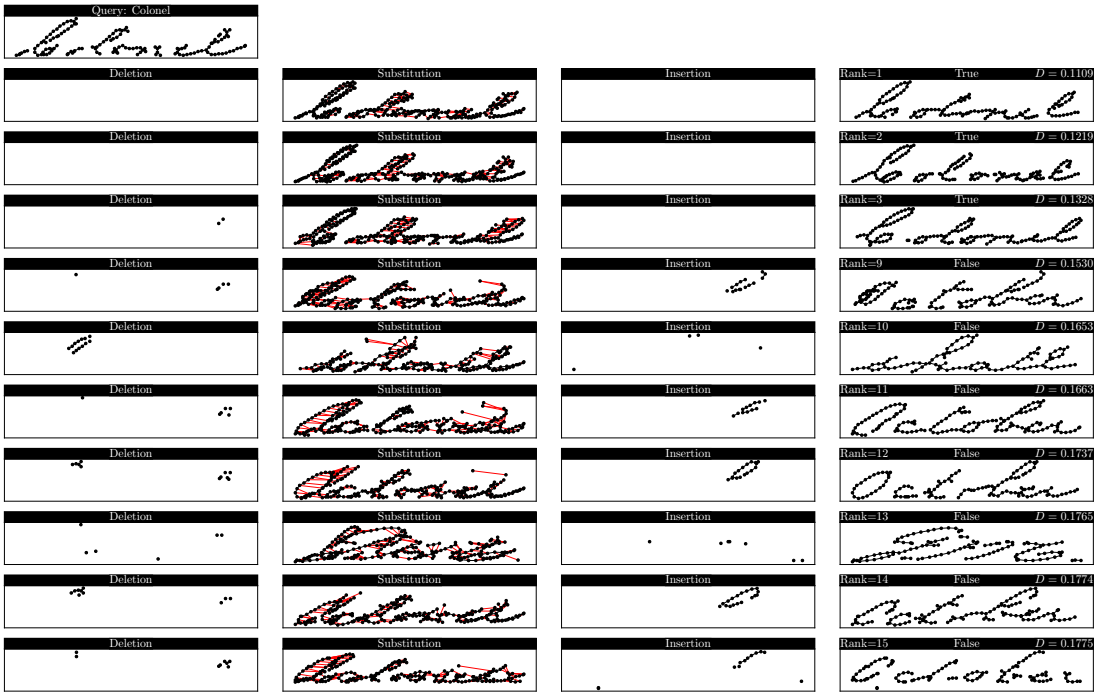


FIGURE A.5: Ten samples from HED matching for query *Colonel*.

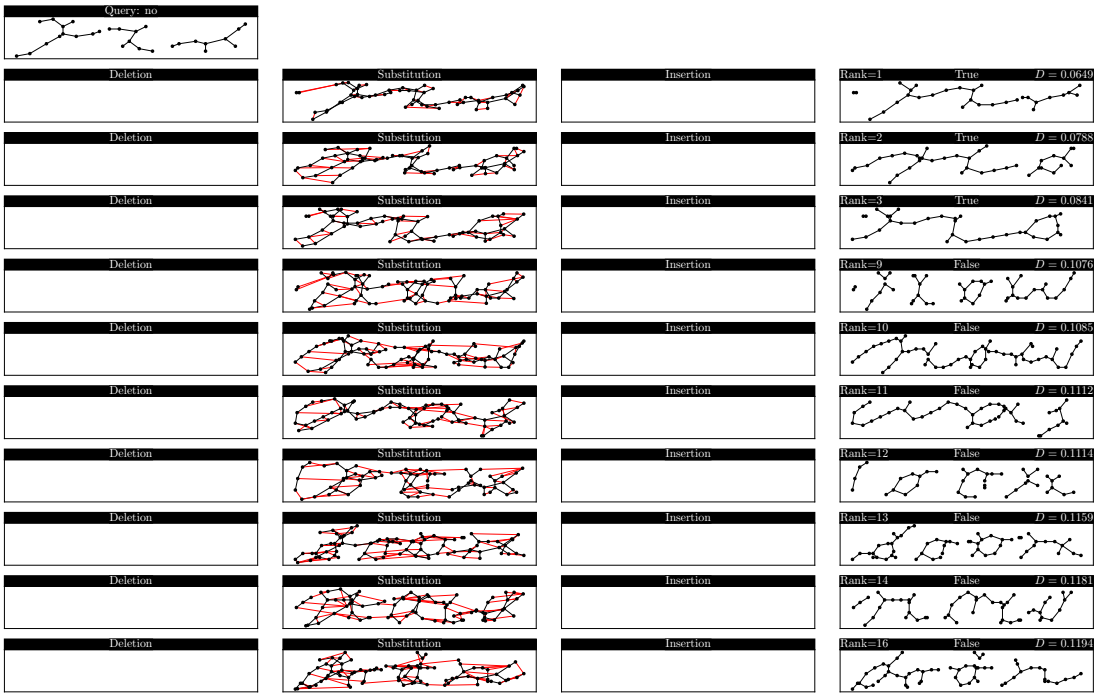


FIGURE A.6: Ten samples from HED matching for query *no*.

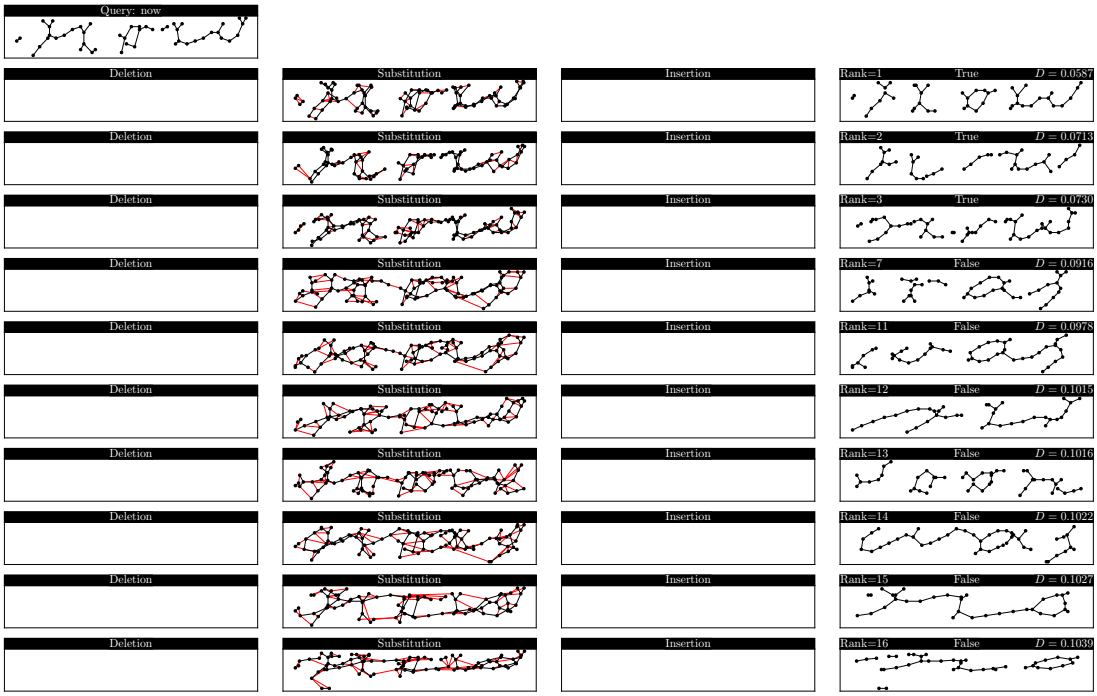


FIGURE A.7: Ten samples from HED matching for query *now*.

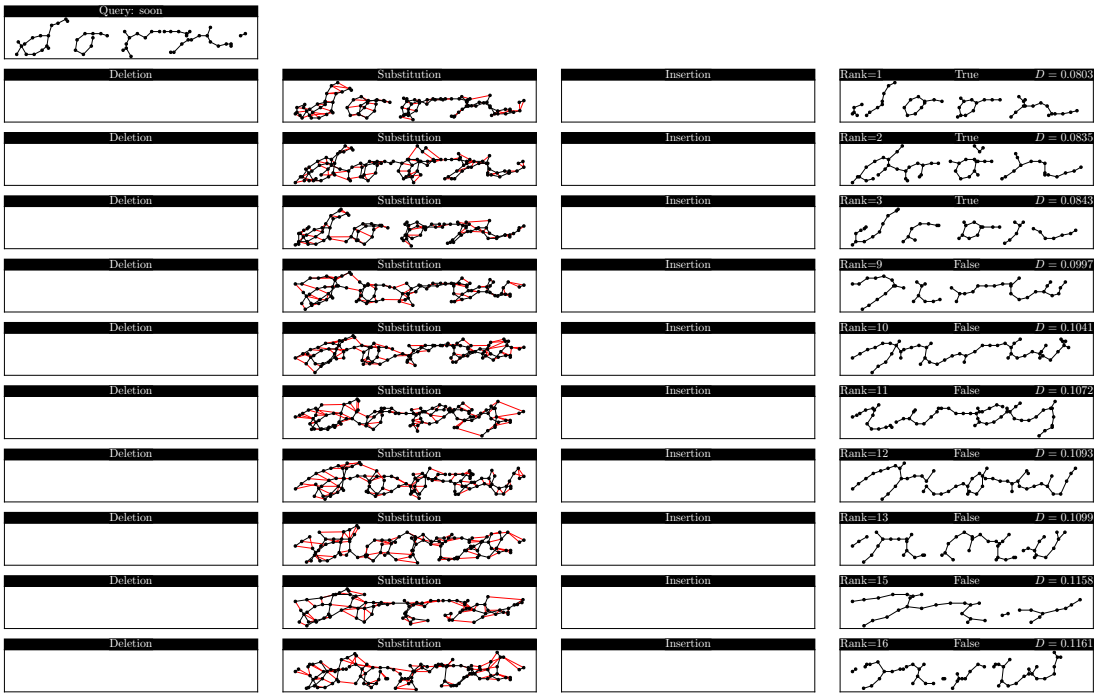


FIGURE A.8: Ten samples from HED matching for query *soon*.

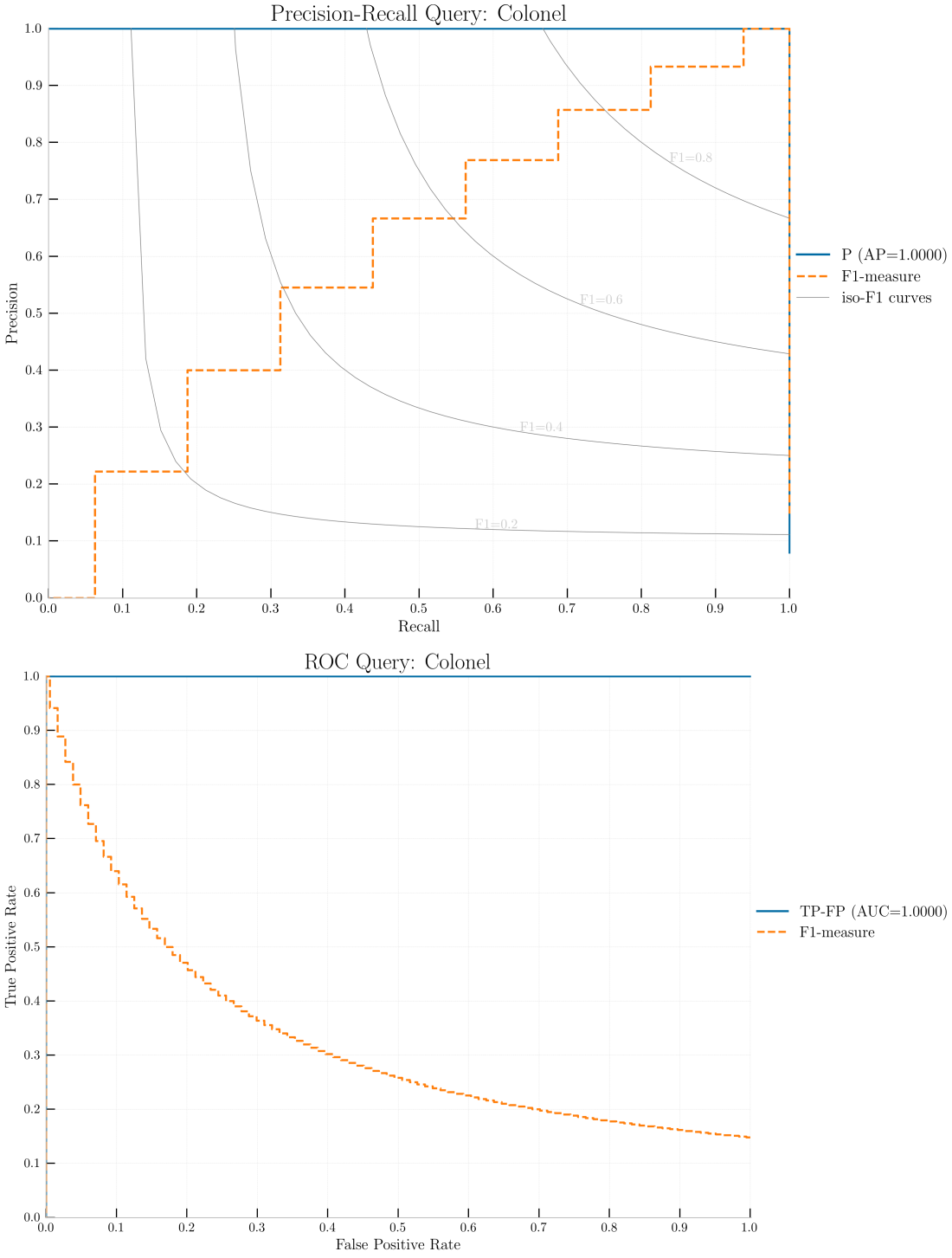


FIGURE A.9: The precision-recall, F1-recall and ROC curves for query *Colonel*.

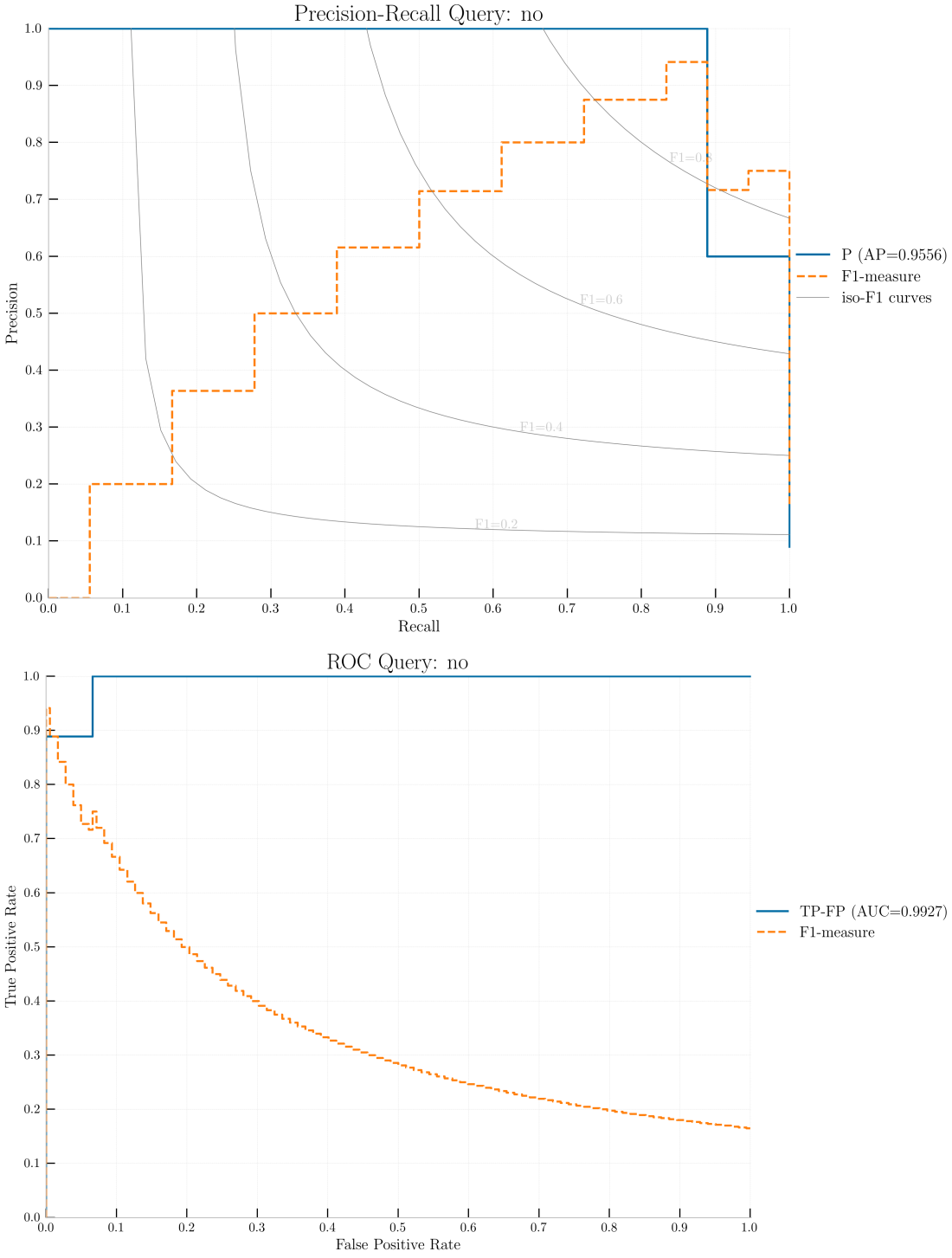


FIGURE A.10: The precision-recall, F1-recall and ROC curves for query *no*.

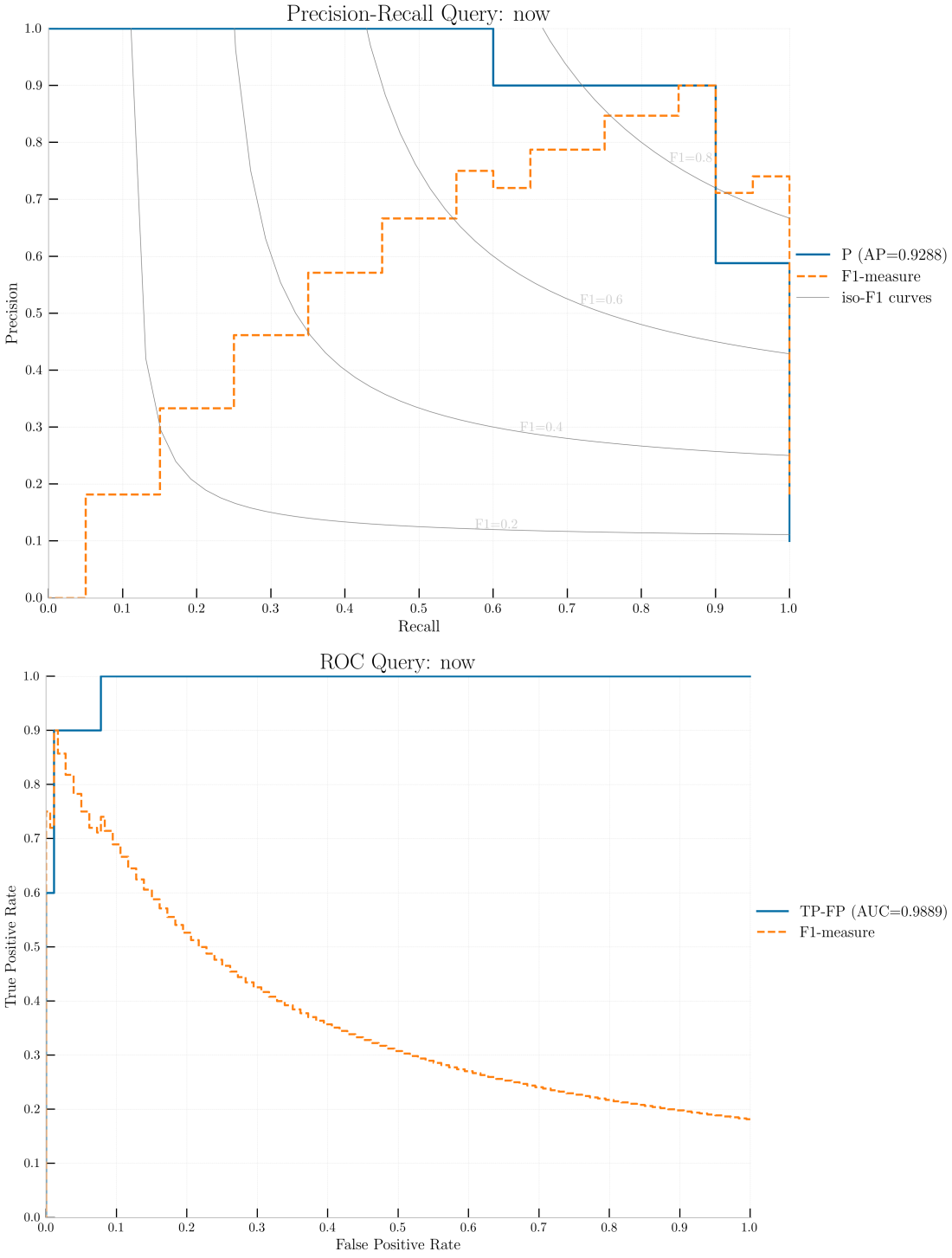


FIGURE A.11: The precision-recall, F1-recall and ROC curves for query *now*.

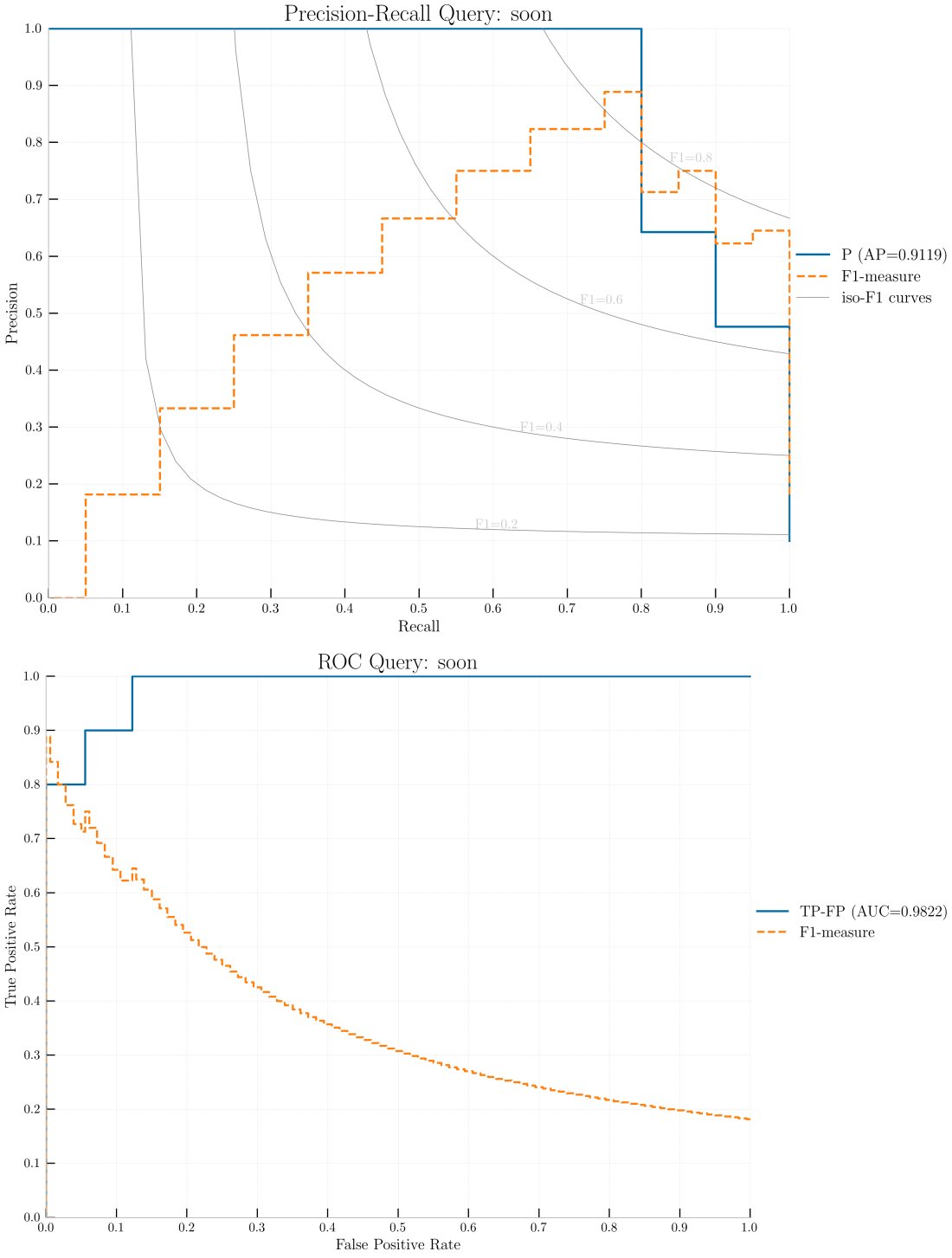


FIGURE A.12: The precision-recall, F1-recall and ROC curves for query *soon*.